Sorting

$$6\ 3\ 9\ 2\ 7\ 1\ 5 \xrightarrow{\text{Sorting}} 1\ 2\ 3\ 5\ 6\ 7\ 9$$

- **Internal sort**: records fit in internal memory (direct access)
  External sort: on external devices — (Block transfer)
  disk, tape ↳ (sequential access)

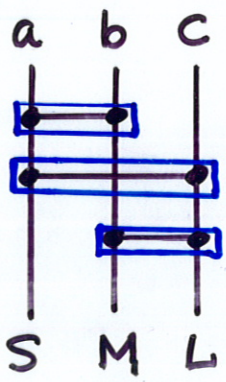
- **T**: $\geq N \log N$ (key comparisons)
  - N^2 : selection, insertion, bubble – elementary
  - $N \log N$: quick (average), heap (worst)
  - N : radix sort, distribution counting

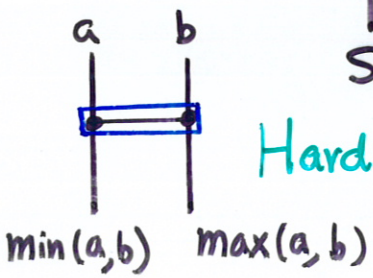
- **S**: in place : NO extra memory

- **Stable** : 2 4 3 1 2 5 1 2 → 1 1 2 2 2 3 4 5
  Unstable : → 1 1 2 2 2 3 4 5

- **3sort**

```
if a > b then swap(a, b)
if a > c then swap(a, c)
if b > c then swap(b, c)
```





Hardware implementation

- **Selection sort**

```
S O R T I N G
G O R T I N S
G I R T O N S
G I N T O R S
G I N O T R S
G I N O R T S
G I N O R S T
```

```
for i := 1 to N-1 do {
    min := i;
    for j := i+1 to N do
        if a[j] < a[min] the
            min := j;
    Swap(a[i], a[min]);
}
```

$$T = \frac{N^2}{2} + NM$$ ← record size

- Good for small N, large M

- **Insertion sort**

```
    S O R T I N G
    O S R T I N G
    O R S T I N G
        ⋮
    I O R S T N G
-∞  I N O R S T G
    G I N O R S T
```

$$T = \frac{N^2}{4} + \frac{N^2}{8}M$$

- For almost sorted file.
- Compare adjacent pairs only ⇓ longer leap

• Shell sort

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E	
															h=13
A	E	O	R	T	I	N	G	E	X	A	M	P	L	S	
		↓				↓				↓				↓	h=4
A	E	A	G	E	I	N	M	P	L	O	R	T	X	S	
															h=1 ← insertion sort
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X	

h-sequence: ..., 1093, 364, 121, 40, 13, 4, 1

• Bubble sort

```
for i := N downto 1
    for j := 2 to i do
        if a[j-1] > a[j] then
            swap(a[j-1], a[j])
```

6	3	3	2	2	1
3	6	2	3	1	2
9	2	6	1	3	3
2	7	1	5	5	5
7	1	5	6	6	6
1	5	7	7	7	7
5	9	9	9	9	9

$$T = \frac{N^2}{2} + \frac{N^2}{2} M$$

Distribution Counting

- Key $\in \{0, 1, \cdots, M-1\}$ $\qquad M = 8$

$i \rightarrow$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$a[i] \rightarrow$	3	1	1	0	3	7	5	5	2	4	2	1	0	2	6	4

	0	1	2	3	4	5	6	7
Count[i]	2	3	3	2	2	2	1	1

	0	1	2	3	4	5	6	7
count[i]	2	5	8	10	12	14	15	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
b[i]	0	0	1	1	1	2	2	2	3	3	4	4	5	5	6	7

```
for i := N downto 1 do {
    b[count[a[i]]] := a[i];
    Count[a[i]] := Count[a[i]] - 1;
}
```

- $T = 2N$
- Stable

Quick Sort

```
Quicksort(l, r)
if r > l then {
    v := a[r];  i := l-1;  j := r;
    repeat
        repeat i := i+1 until a[i] ≧ v;
        repeat j := j-1 until a[j] ≦ v;
        t := a[i];  a[i] := a[j];  a[j] := t;
    until j ≦ i;
    a[j] := a[i];  a[i] := a[r];  a[r] := t;
    quicksort(l, i-1);
    quicksort(i+1, r);
}
```

$i :=$ partition(l, r);

← Until $j < i$;

$a[r] \leftrightarrow a[i]$

← quicksort(l, j);

$i :=$ Partition(l, r)

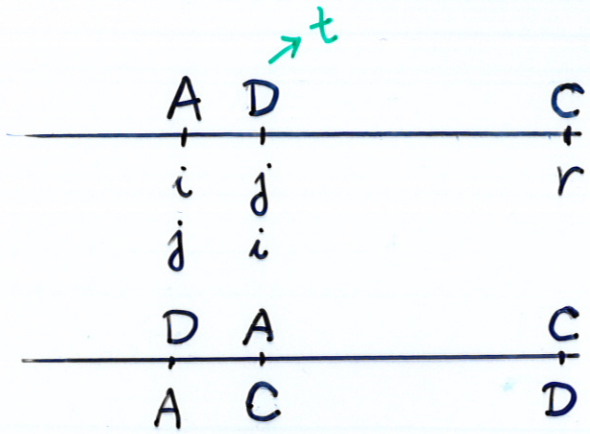
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$-\infty$	A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
	A	A									S	M	P	L	E
	A	A	E						O	X	S	M	P	L	E
	A	A	E	E	T	I	N	G	O	X	S	M	P	L	R

sentinel

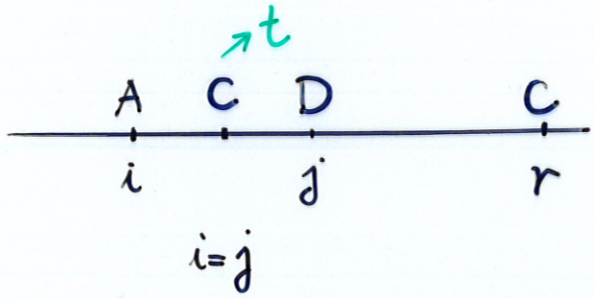
$\leq a[i]$ $\quad$ $a[i]$ $\quad$ $\geq a[i]$

- Inefficiently on sorted file.
- $a[i] \geq v$, $a[j] \leq v$ $\Rightarrow$
  (1) Sentinel
  (2) balance partition
  (3) less complicated

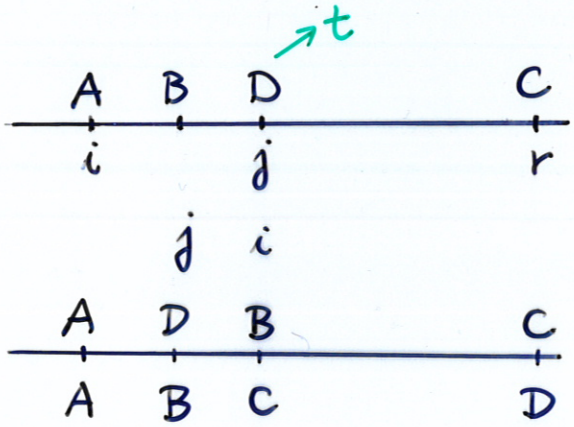
(1)



(2)



(3)



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	$\boxed{E}$	T	I	N	G	O	X	S	M	P	L	R
A	A	$\boxed{E}$												
$\boxed{A}$	A													
	$\boxed{A}$													
				L	I	N	G	O	P	M	$\boxed{R}$	X	T	S
				L	I	G	$\boxed{M}$	O	P	N				
				$\boxed{G}$	I	L								
					I	$\boxed{L}$								
					$\boxed{I}$									
								$\boxed{N}$	P	O				
									$\boxed{O}$	P				
										$\boxed{P}$				
												$\boxed{S}$	T	X
													T	$\boxed{X}$
													$\boxed{T}$	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

- $$C_N = \frac{1}{N}\sum_{1\le k\le N}\left(N+1+C_{k-1}+C_{N-k}\right) \qquad C_0 = C_1 = 0$$

$$= N+1+\frac{2}{N}\sum_{1\le k\le N-1} C_k$$

$$C_{N-1} = N + \frac{2}{N-1}\sum_{1\le k\le N-2} C_k$$

$$\therefore\ C_N = \frac{N+1}{N}C_{N-1} + 2$$

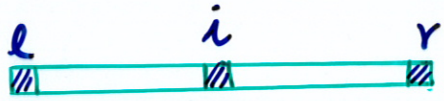
$$\vdots$$

$$C_N = (N+1)\left(1+\frac{2}{4}+\frac{2}{5}+\cdots+\frac{2}{N+1}\right)$$

$$\approx 2N\lg N$$

Improvements (25% ~ 30%)

- **Removing recursion:**




```
l := 1;   r := N;   initStack;  push(l);  push(r);
repeat
   if r > l then {
        i := partition(l, r);
        if i-l > r-i then                    ← put larger subfile
              push(l); push(i-1);  l := i+1;   on stack
        else  push(i+1); push(r);  r := i-1;
   else
        r := pop;   l := pop;
until Stack = ϕ
```

lgN entries on stack

- **Small subfile**

  if $r - l \le 15$ then insertion(l, r);

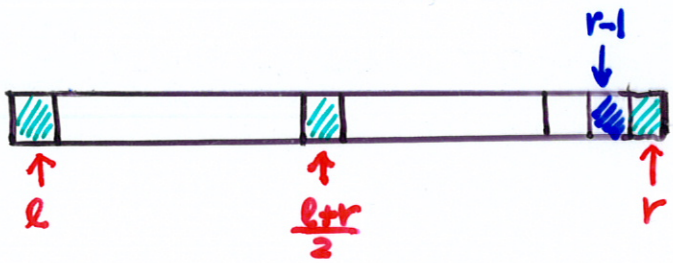
- **Median-of-three**

  threeSort$(a[l],\ a[\frac{l+r}{2}],\ a[r])$;

  swap$(a[\frac{l+r}{2}],\ a[r-1])$;

  partition$(l+1,\ r-2)$;

  ⋮



- No sentinel.
- partition$(l+1, r-2)$
- $i \approx \frac{l+r}{2}$

Selection : Find Kth Smallest ($K=\frac{N}{2}$, median)

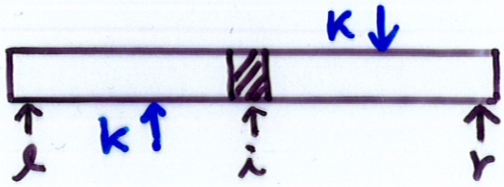
- K small : $(\text{selection sort})^K$ $\quad T = NK$
- K larger : priority queue : $(\text{insert})^K (\text{replace})^{N-K}$ $\quad T = N \log K$
- K Large :

Recursive

```
Select(l, r, K)
  if r > l then {
      i := partition(l, r);
      if i > l+K-1 then
          select(l, i-1, K);
      if i < l+K-1 then
          Select(i+1, r, K-i);
  }
```

Nonrecursive K: global

```
Select(K)   l:=1;  r:=N;
  While r > l do {
      i := partition(l, r);
      if i ≥ K then
          r := i-1;
      if i ≤ K then
          l := i+1;
  }
```



Example:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
							M							

$T = N^2$ Worst

$N + K \log \frac{N}{K}$ Average

Radix Sorting (string sorting)

- Keys represented in base-M system (M進位) (M : radix)
- Process + compare pieces of keys vs. compare + exchange

$$\text{bits}(x, k, j) := (x \text{ div } 2^k) \bmod 2^j$$

- $T = Nb \approx N \log N$

(1) Radix Exchange sort (L → R)

M=2

A 00001	A 00001	A 00001	A 00001	A 00001	A 00001
S 10011	E 00101	E 00101	A 00001	A 00001	A 00001
O 01111	O 01111	A 00001	E 00101	E 00101	E 00101
R 10010	L 01100	E 00101	E 00101	E 00101	E 00101
T 10100	M 01101	G 00111	G 00111	G 00111	
I 01001	I 01001	I 01001	I 01001		
N 01110	N 01110	N 01110	N 01110	L 01100	L 01100
G 00111	G 00111	M 01101	M 01101	M 01101	M 01101
E 00101	E 00101	L 01100	L 01100	N 01110	N 01110
X 11000	A 00001	O 01111	O 01111	O 01111	O 01111
A 00001	X 11000	S 10011	S 10011	P 10000	
M 01101	T 10100	T 10100	R 10010	R 10010	R 10010
P 10000	P 10000	P 10000	P 10000	S 10011	S 10011
L 01100	R 10010	R 10010	T 10100		
E 00101	S 10011	X 11000			

2) Straight Radix sort (R → L)

124, 324, 223, 321, 123, 131

$M = 2^m = 4$

1 → 321, 131
2 →
3 → 223, 123
4 → 124, 324

321, 131, 223, 123, 124, 324

1 →
2 → 321, 223, 123, 124, 324
3 → 131
4 →

321, 223, 123, 124, 324, 131

1 → 123, 124, 131
2 → 223
3 → 321, 324
4 →

123, 124, 131, 223, 321, 324

- Each pass ensures stable
  uses distribution counting

- Example:

$m = 1$

A 00001	R 10010	T 10100	X 11000	P 10000	A 00001
S 10011	T 10100	X 11000	P 10000	A 00001	A 00001
O 01111	N 01110	P 10000	A 00001	A 00001	E 00101
R 10010	X 11000	L 01100	I 01001	R 10010	E 00101
T 10100	P 10000	A 00001	A 00001	S 10011	G 00111
I 01001	L 01100	I 01001	R 10010	T 10100	I 01001
N 01110	A 00001	E 00101	S 10011	E 00101	L 01100
G 00111	S 10011	A 00001	T 10100	E 00101	M 01101
E 00101	O 01111	M 01101	L 01100	G 00111	N 01110
X 11000	I 01001	E 00101	E 00101	X 11000	O 01111
A 00001	G 00111	R 10010	M 01101	I 01001	P 10000
M 01101	E 00101	N 01110	E 00101	L 01100	R 10010
P 10000	A 00001	S 10011	N 01110	M 01101	S 10011
L 01100	M 01101	O 01111	O 01111	N 01110	T 10100
E 00101	E 00101	G 00111	G 00111	O 01111	X 11000

3) Linear sort

(i) 4 passes straight radix sort $(m = \frac{b}{4})$ ⇒ Linear

$T = 4N$

(ii) 2 passes on leading bits (almost sorted now)
+ insertion sort

Priority Queue

- Example : Job Scheduler, Key = Priority

- Operations :

  Construct $= (\text{insert})^N$

  * Insert

  * Remove largest element (unless v is larger)

  Replace(v) largest with new item v = insert + remove

  Change priority of an element = delete + insert

  Join 2 queues

  Delete(v)

- Priority queue for Sorting $= (\text{insert})^N (\text{remove})^N$

1) unordered list $\Rightarrow$ Selection Sort : burden on remove

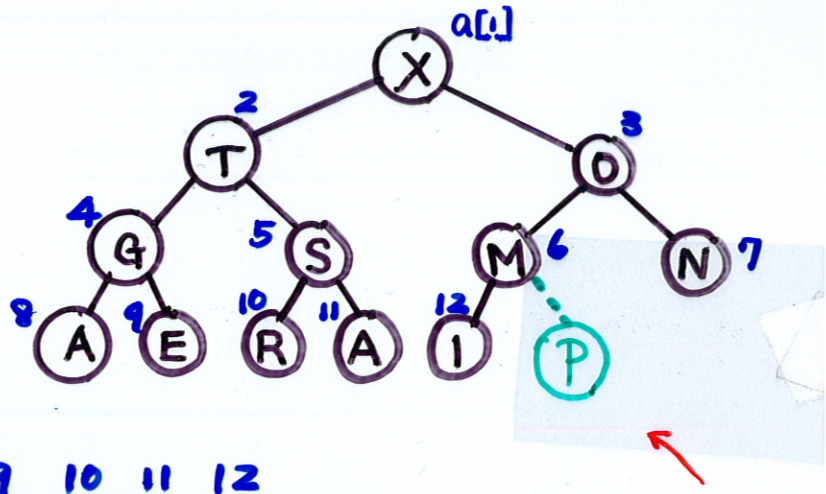
2) ordered list $\Rightarrow$ Insertion Sort : " insert

3) <u>Heap</u> $\Rightarrow$ <u>Heap sort</u> : amortize

$T = N \log N$

• Heap

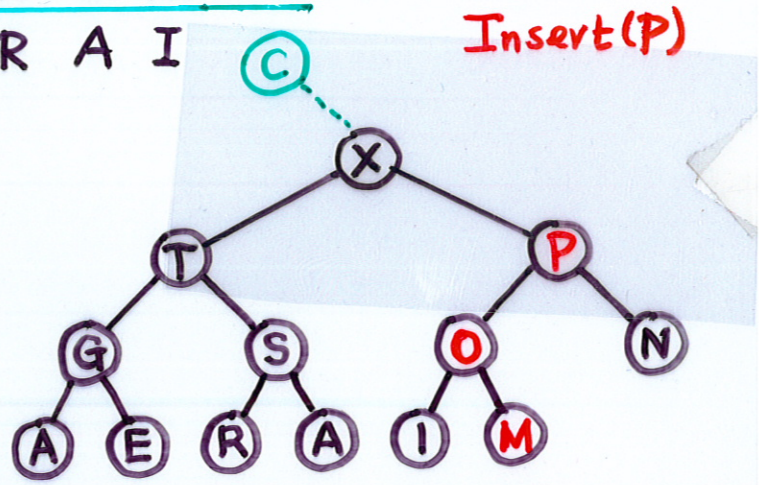
$a[j] \geq a[2j]$
$\geq a[2j+1]$



i	1	2	3	4	5	6	7	8	9	10	11	12
a[i]	X	T	O	G	S	M	N	A	E	R	A	I

j div 2

Insert(P)

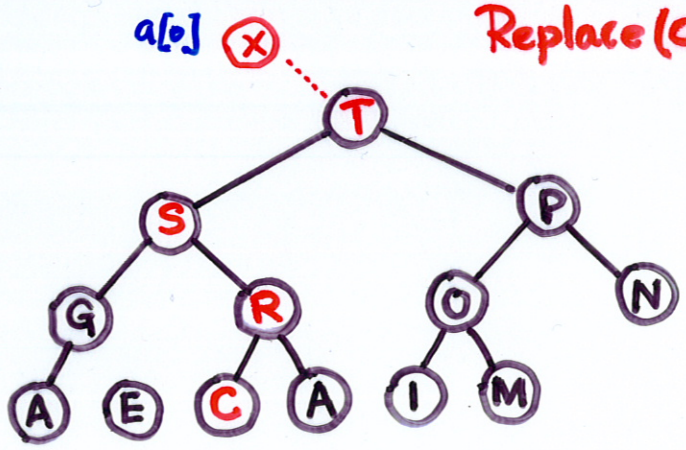


• Upheap(k)

• Insert(v)

N := N+1; a[N] := v;
upheap(N);

Replace(C)



• Downheap(k)

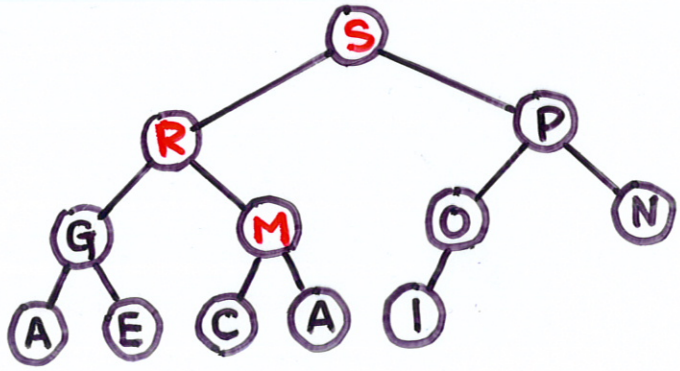
• Replace(v)

a[0] := v; downheap(0);
replace := a[0];

• Remove

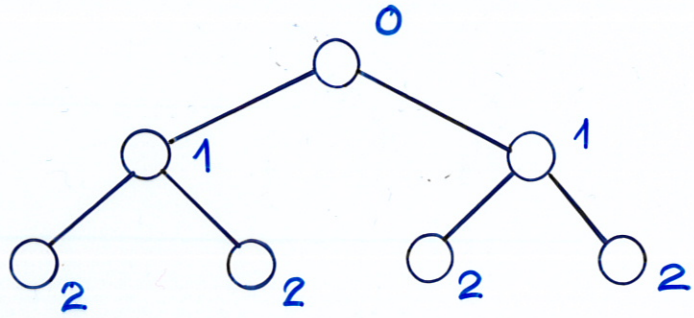
remove := a[1];
a[1] := a[N]; N := N-1;
downheap(1);

Remove

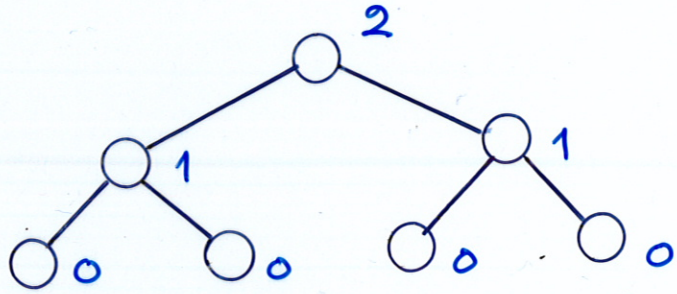


$T = O(\log N)$

Top-down:



Bottom-up:



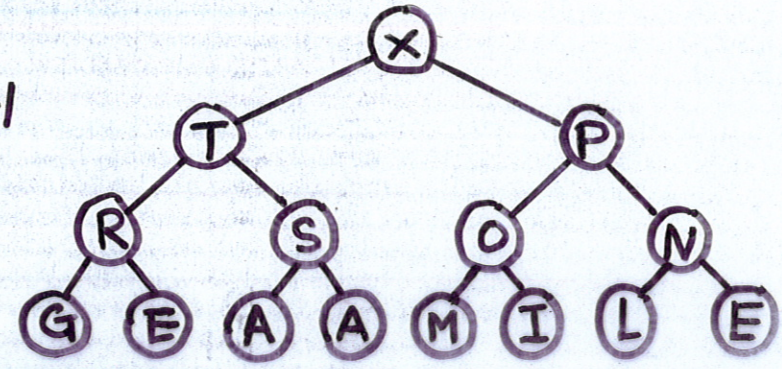
- **Heap sort**

(1) $(\text{insert})^M(\text{remove})^M$: Top down

(2)

```
N := M;
for k := M/2 downto 1 do
    downheap(k);
repeat
    swap(a[1], a[N]); N := N-1
    downheap(1)
until N ≤ 1
```

} Construct Heap : Bottom up



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
						N							L	E
					P						M	I		
				X					T	A				
			R				G	E						
		P			O	N					M	I	L	E
	X		R	T			G	E	S	A				
X	T	P	R	S	O	N	G	E	A	A	M	I	L	E
T	S	P	R	E	O	N	G	E	A	A	M	I	L	
S	R	P	L	E	O	N	G	E	A	A	M	I		
R	L	P	I	E	O	N	G	E	A	A	M			
P	L	O	I	E	M	N	G	E	A	A				
O	L	N	I	E	M	A	G	E	A					
N	L	M	I	E	A	A	G	E						
M	L	E	I	E	A	A	G							
L	I	E	G	E	A	A								
I	G	E	A	E	A									
G	E	E	A	A										
E	A	E	A											
E	A	A												
A	A													
A														
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Selection & Merging

- **Selection**: find Kth smallest
  median ($K = \frac{N}{2}$)
  split file (divide + Conquer)

- **Merging**: sorted file + sorted file $\hookrightarrow$ larger sorted file
  merge file

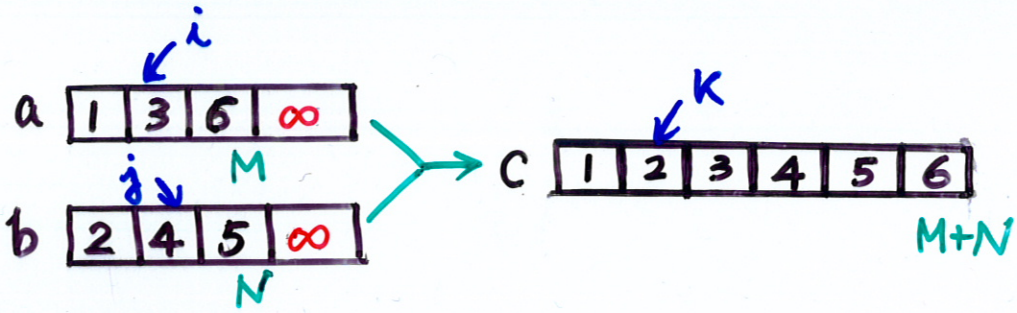
Quick Sort: divide 2 sorted file $\Rightarrow$ whole file sorted
: most done before recursive calls (Conquer & Divide)

Merge Sort: " + merge $\Rightarrow$ "
: most done after recursive calls (Divide & Conquer)

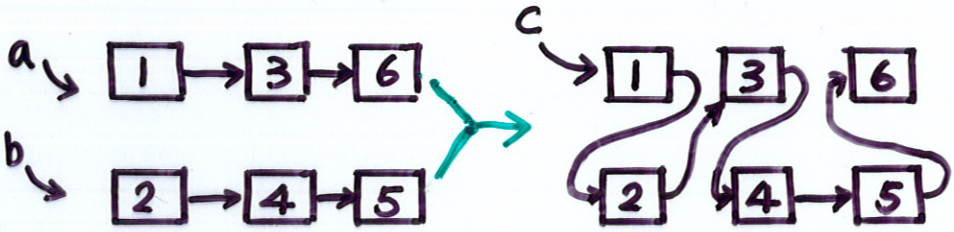
• Merge

2-Way merge:

array merge:



list merge:



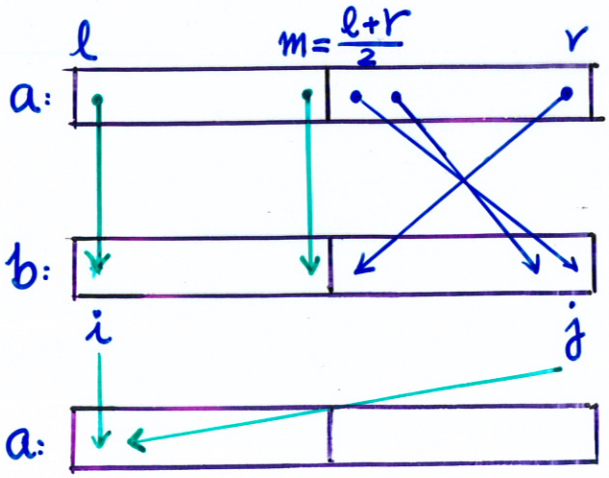
$T = M + N$

```
i := 1;  j := 1;
a[M+1] := ∞;  b[N+1] := ∞;
for K := 1 to M+N do
    if a[i] < b[j] then c[K] := a[i];  i := i+1;
                   else c[K] := b[j];  j := j+1;
```

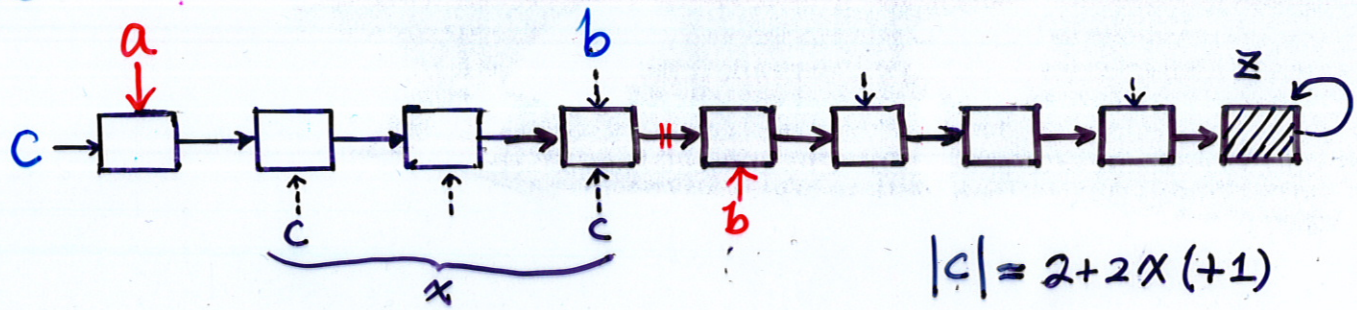
• Merge Sort

- array

```
m := (l+r) div 2;
mergeSort(l, m);  mergeSort(m+1, r);
Copy(a, b, l, m);  CopyR(a, b, m+1, r);
merge
```



- List



$|c| = 2 + 2x \ (+1)$

```
a := c;  b := c.next;  b := b.next;  b := b.next;
while b ≠ z do
    c := c.next;  b := b.next;  b := b.next;
b := c.next;  c.next := z;
mergeSort := Merge (mergeSort (a), mergeSort (b));
```

- Build up small sorted file into larger ones (top down)
- $M_N = 2M_{\frac{N}{2}} + N$
  $= N \lg N$
- Example:

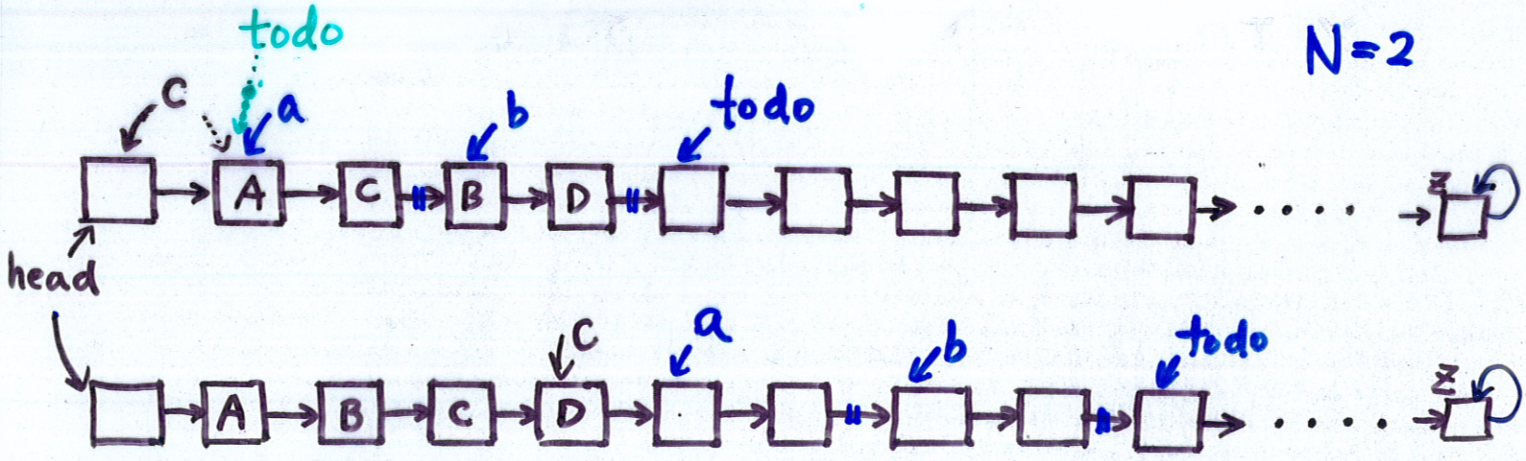
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
	O	S												
A	O	S												
			R	T										
					I	N								
			I	N	R	T								
A	I	N	O	R	S	T								
							E	G						
									A	X				
							A	E	G	X				
											M	P		
													E	L
											E	L	M	P
							A	E	E	G	L	M	P	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Merge Sort (Bottom up)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N=1	A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
2	A	S	O	R	I	T	G	N	E	X	A	M	L	P	E
4	A	O	R	S	G	I	N	T	A	E	M	X	E	L	P
8	A	G	I	N	O	R	S	T	A	E	E	L	M	P	X
16	A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

```
begin
N:=1; new(head); head↑.next:=c;
repeat
  todo:=head↑.next; c:=head;
  repeat
    t:=todo;
    a:=t; for i:=1 to N−1 do t:=t↑.next;
    b:=t↑.next; t↑.next:=z;
    t:=b; for i:=1 to N−1 do t:=t↑.next;
    todo:=t↑.next; t↑.next:=z;
    c↑.next:=merge(a, b);
    for i:=1 to N+N do c:=c↑.next
  until todo=z;
  N:=N+N;
until a=head↑.next;
mergesort:=head↑.next
end;
```

$N+N \rightarrow 2N$



N=4

- Stable

No stack

External Sorting

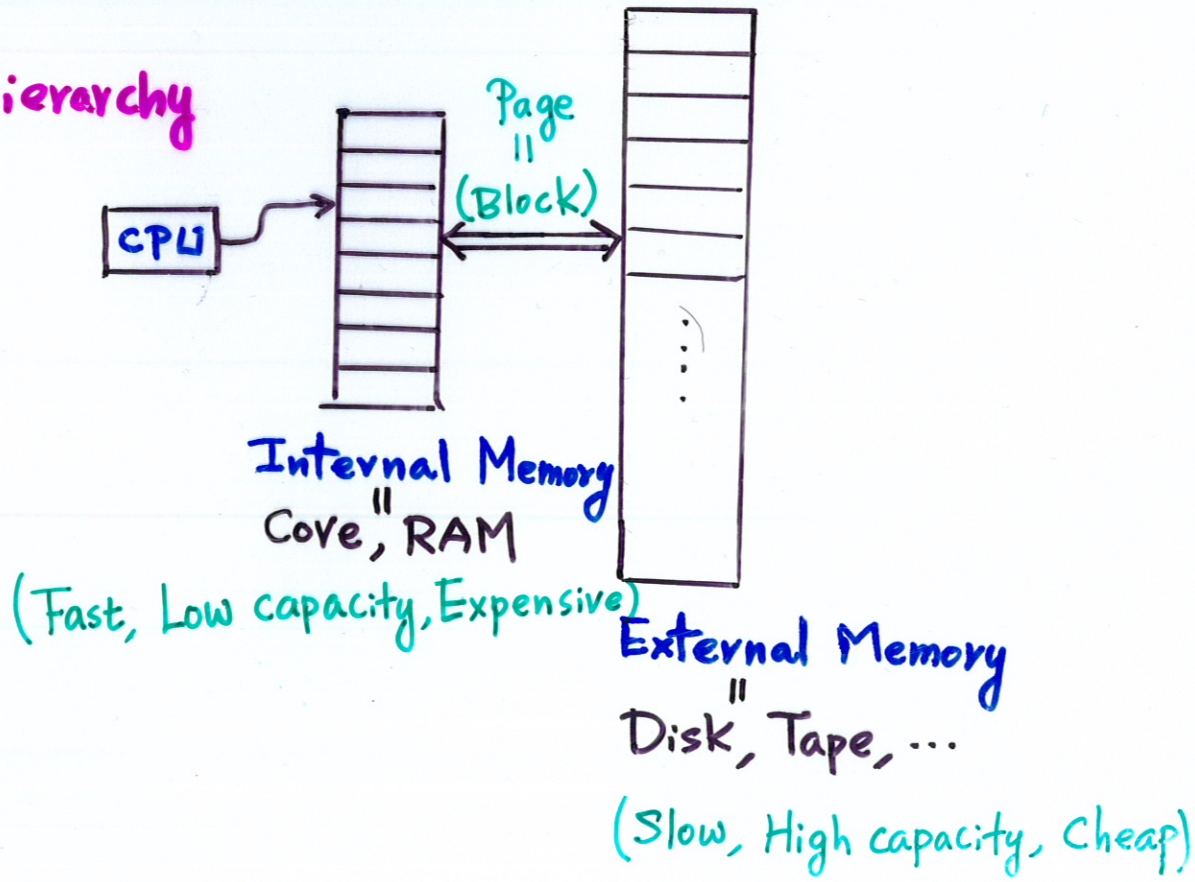
- External memory : disk, tape, ...
- Accessing cost dominates
- System aspect as important as algorithm (technology dep.)
- 2-level memory hierarchy

- **General Sort-Merge method:**
  - Break file into sorted blocks (runs)
    + Merge into larger block

- **Goal:**

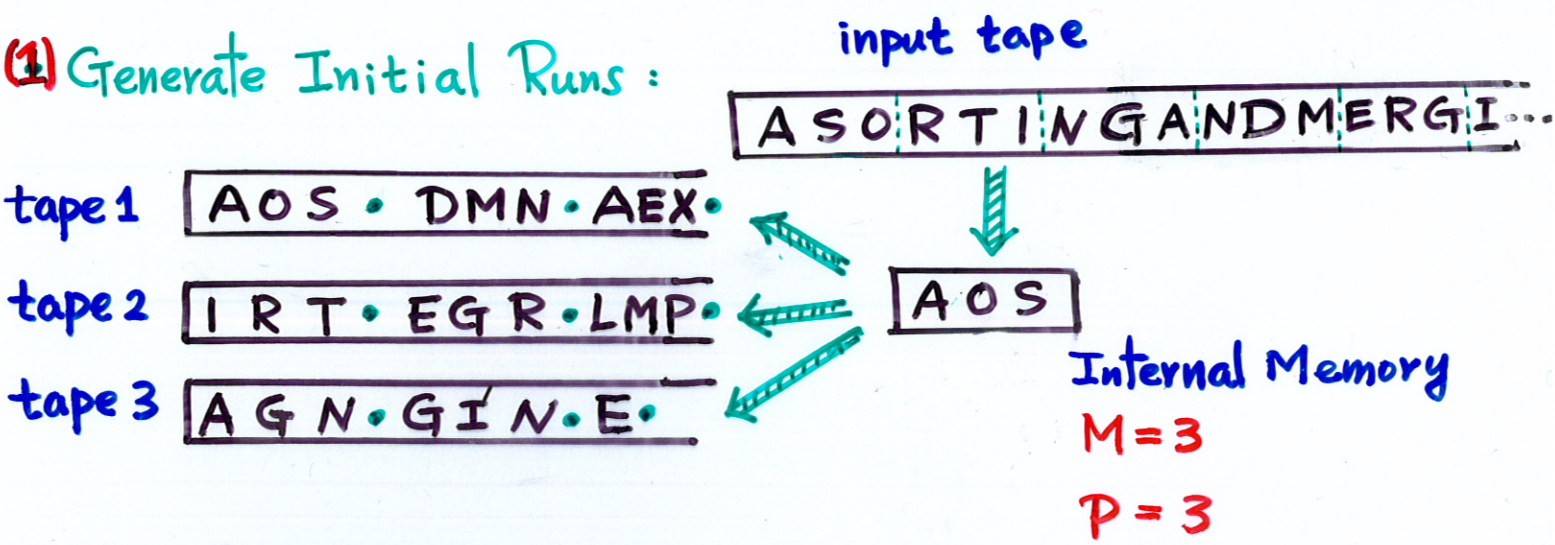
  Reduce # passes

- **Memory Hierarchy**

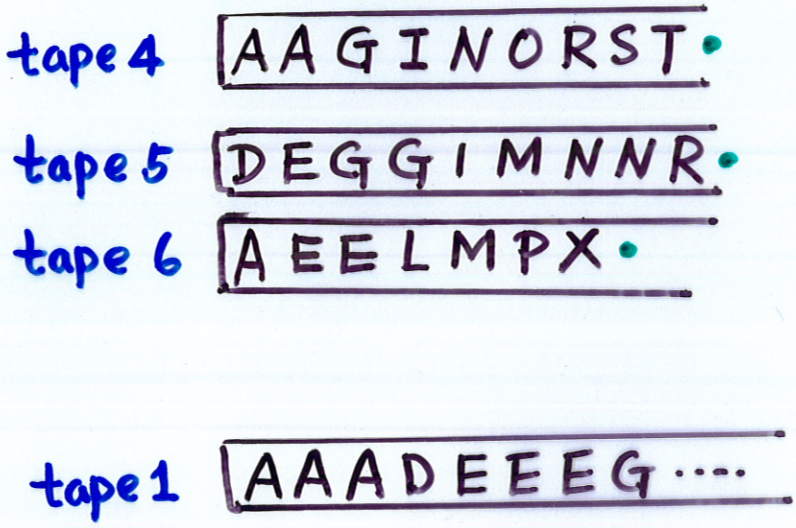


· Balanced P-way merge

(1) Generate Initial Runs :



(2) P-way Merge



- $\log_P \frac{N}{M}$ passes
- 4-tapes are enough
- How to merge ?

- **Replacement Selection**

Use Priority Queue

- For P-way merge:

tape1: A O S • D M N • • • •

tape2: I R T • E G R • • • •

tape3: A G N • G I N • • • •

tape4: A A G I N O R S T •

Heap, size P

1	2	3
A	I	A
A	I	O
G	I	O
I	N	O
N	R	O
O	R	• ↔ ∞
R	S	•
S	T	•
T	•	•

- Form Initial Runs

Heap. size = M

R	A	S	O
T	O	S	R
I	R	S	T
N	S	I*	T
G	T	I*	N*
A	G*	I*	N*
N	I*	A	N*
D	N*	A	N*
M	N*	A	D

E	A	M	D
R	D	M	E
G	E	M	R
I	G	M	R
N	I	M	R
G	M	N	R
E	N	G*	R
X	R	G*	E*
A	X	G*	E*

M	A*	G*	E*
P	E*	G*	M*
L	G*	P*	M*
E	L*	P*	M*
	M*	P*	E
	P*		E
	E		

tape1: A O R S T • A D E G I M N R X • E •

tape2: G I N N • A E G L M P •

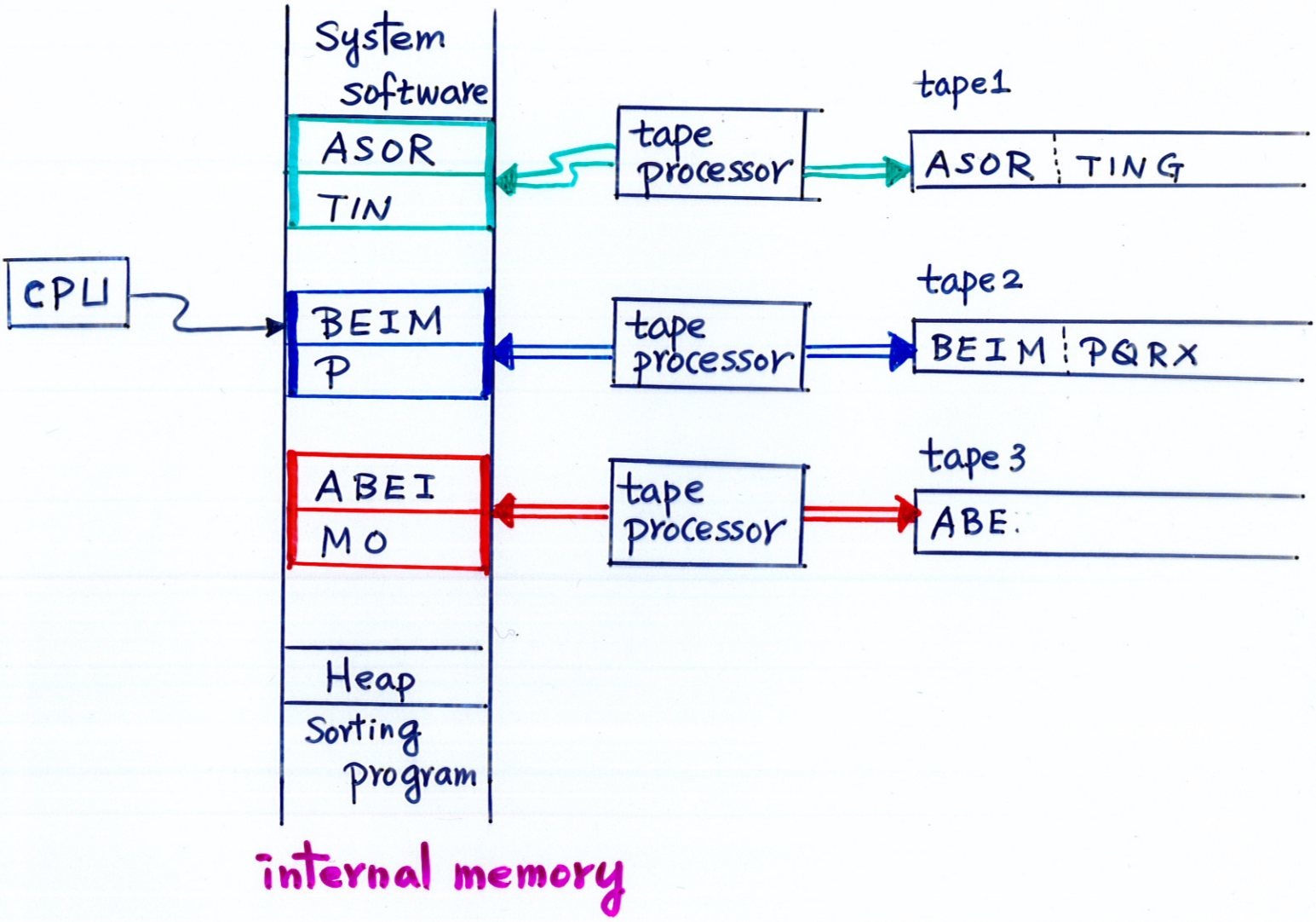
- Random Keys ⇒ Run Size ≐ 2M

  Some order ⇒ Longer runs

  (Example: ∀ key, less than M larger key before it ⇒ 1 run)

- M P tapes ⇒ $\log_P \frac{N}{2M}$ passes.

- System architecture

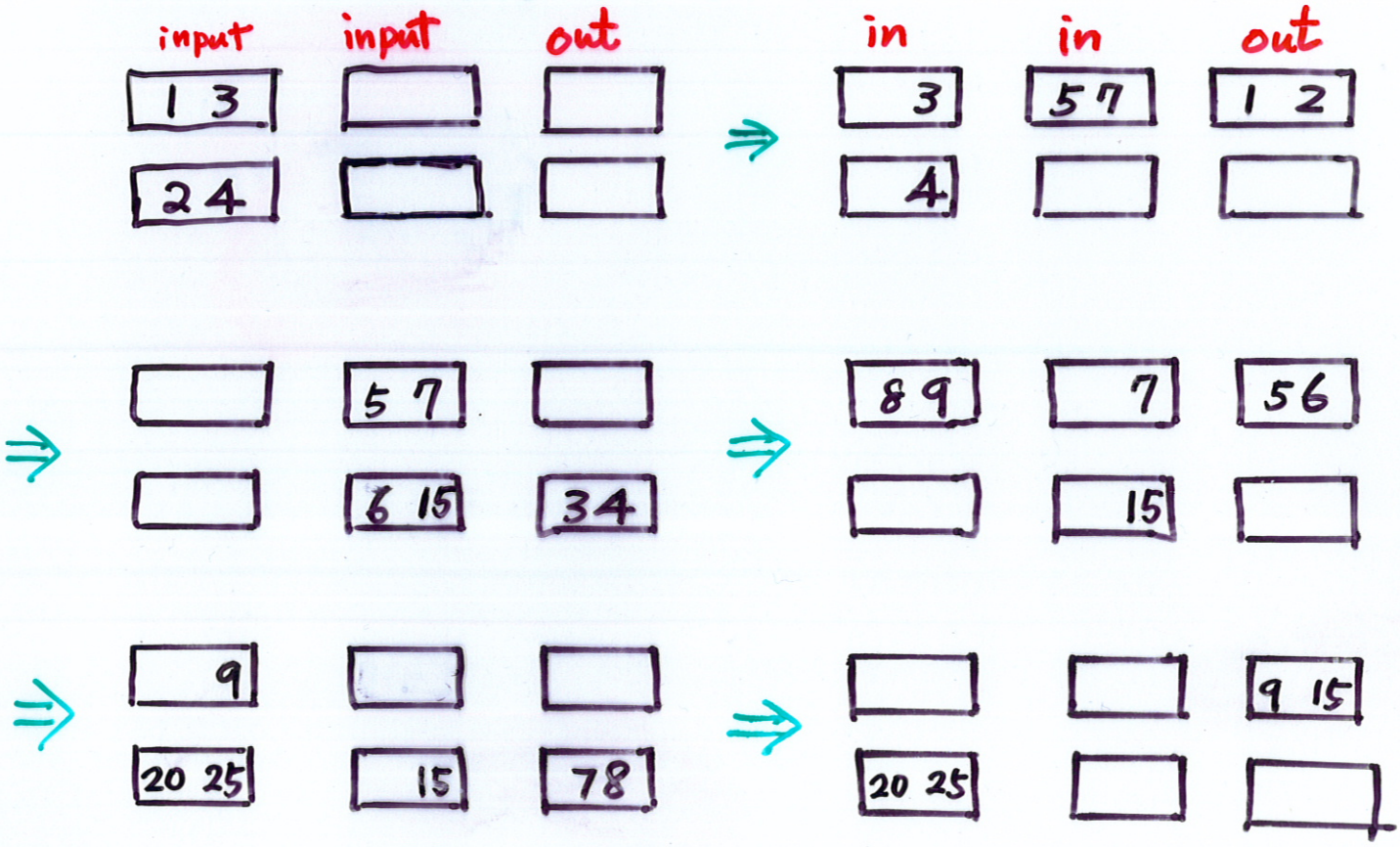


- Practical consideration
  - overlapping : Input, output, computing
  - double buffering
  - Forecasting

Practical Considerations

- Overlap Input, Output, & Computing
- Double Buffering : P-way merging 2P : input buffers
  2 : output "

tape1: 1, 3, 5, 7, 8, 9, ...
tape2: 2, 4, 6, 15, 20, 25, ·
tape3: 1, 2, 3, 4, ...



- Forecasting P+1 input buffers
  2 output "

Example: 3 < 4, ∴ [57]

• Polyphase merging

- No need to i) use 2P tapes
  OR ii) copy files between merging phases.

Example:

tape1: AORST · IN · AGN · DEMR · GIN ·
tape2: EGX · AMP · EL ·
tape3:

---

tape1: DEMR · GIN ·
tape2:
tape3: AEGORSTX · AIMNP · AEGLN ·

---

tape1:
tape2: ADEEGMORRSTX · AGIIMNNP ·
tape3: AEGLN ·

---

tape1: AADE ..... (sorted file)
tape2:
tape3:

How to distribute initial 31 runs ?

T1	T2	T3	T4
0	13	11	7
7	6	4	0
3	2	0	4
1	0	2	2
0	1	1	1
1	0	0	0

$13 = 7 + 4 + 2$

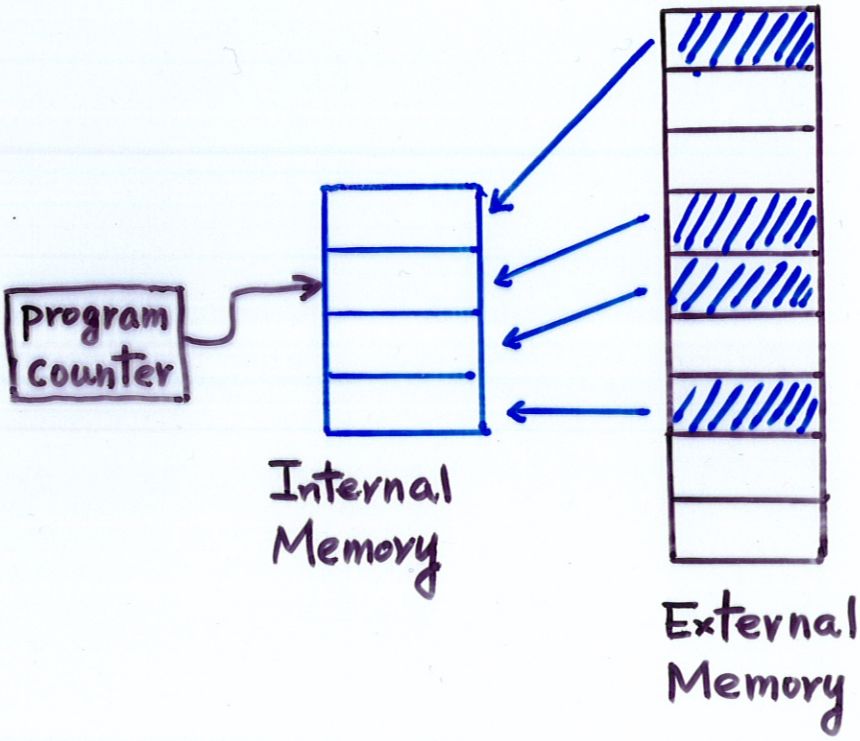
Generalized Fibonanci #s.

Virtual Memory Environment (Easy!)

- Page demanding + Least Recently Used (LRU) page
- Locality of Reference

- Internal Sorting with few locality works well.

Example: Quick sort : 2 localities
Radix sort : 0 "

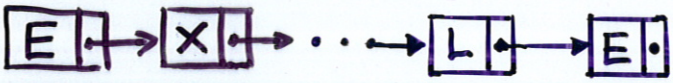


Searching

- retrieve records with Keys matching a given Key.
- dictionary, symbol tables
- Operations: Search
  Insert
  delete
  join
  sort

- **Sequential search:**
  - array: EXAMPLE

    $T = N$: unsuccessful
    $N/2$: successful
  - linked list:



  - most frequently accessed records go first
  - self-organized:

EXAMPLE

⇓ Search M

MEXAPLE

⇓ Search L

LMEXAPE

▲ Assume locality of reference for search keys

• Binary search (sorted file)

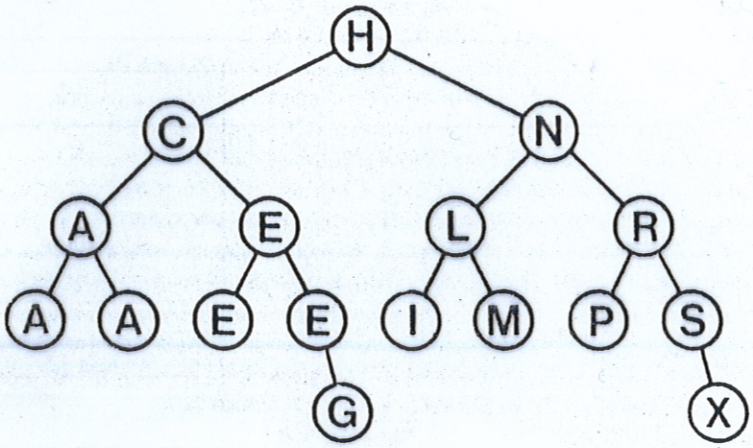
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A	A	A	C	E	E	E	G	H	I	L	M	N	P	R	S	X
A	A	A	C	E	E	E	G	$\boxed{H}$	I	L	M	N	P	R	S	X
									I	L	M	$\boxed{N}$	P	R	S	X
													P	$\boxed{R}$	S	X
															$\boxed{S}$	X

$T = \lg N$

• Binary search Tree (dynamic)

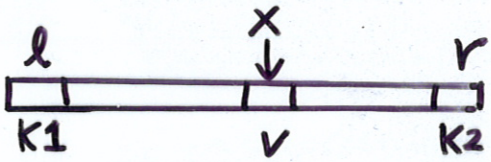
$T = \lg N$ (ave.)

N (worst)



Sort

```
                    root
                     ↓
treeprint(x)                /* in-order traversal */
    if x ≠ nil then {
        treeprint(x↑.left);
        printNode(x);
        treeprint(x↑.right);
    }
```



• Interpolation search

$$x = l + (r - l)\,\frac{v - K1}{K2 - K1}$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A		A	C	E	E	E	G	H	I	L	M	$\boxed{N}$	P	R	S	X
													P	$\boxed{R}$	S	X
															$\boxed{S}$	X

$$13 = 1 + 16 \cdot \frac{S - A}{X - A}$$

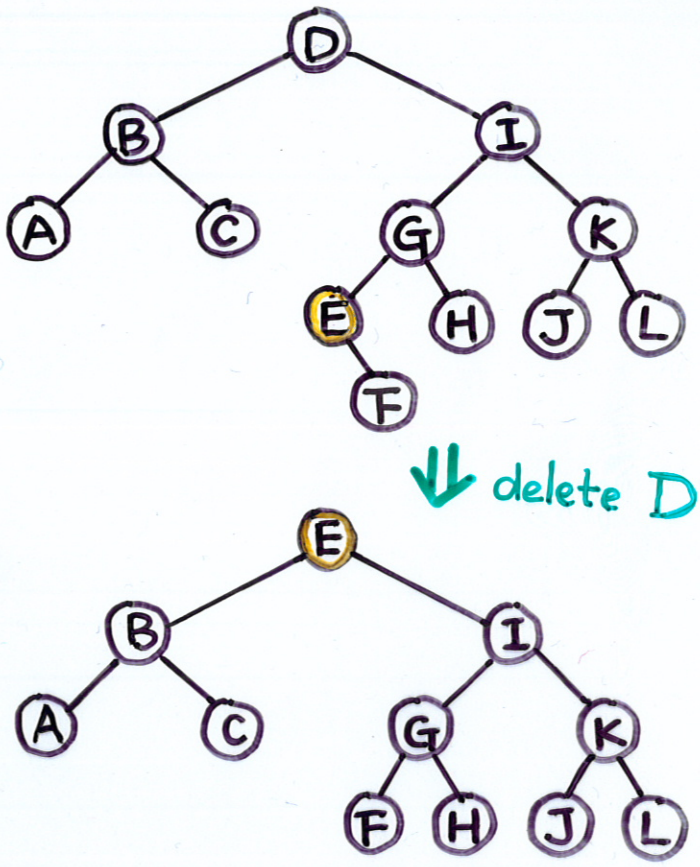
• $T = \lg \lg N \approx$ constant. $(\lg \lg 10^6 < 5)$

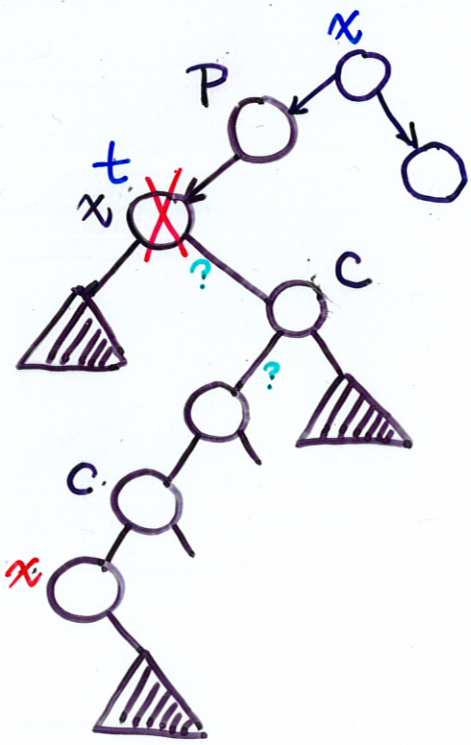
• Deletion

root of tree

```
delete (t, x)
repeat
  p := x;
  if t.key < x.key
      then  x := x.l;
      else  x := x.r;
until t = x;

c := x.r;
if c = z then
    x := x.l;
elsif c.l = z then
    c.l := x.l;
    x := c
else
    while c↑.l↑.l ≠ z do c := c.l;
    x := c.l;
    c.l := x.r;
    x.l := t.l;
    x.r := t.r;
end
if t.key < p.key then p.l := x;
                 else p.r := x;
```



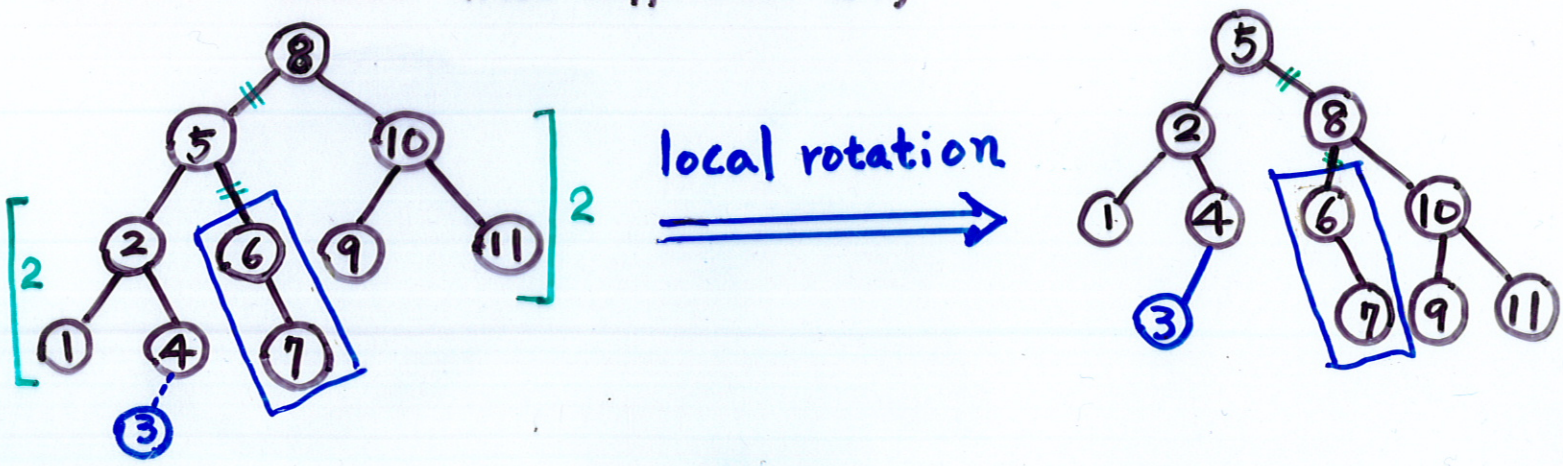


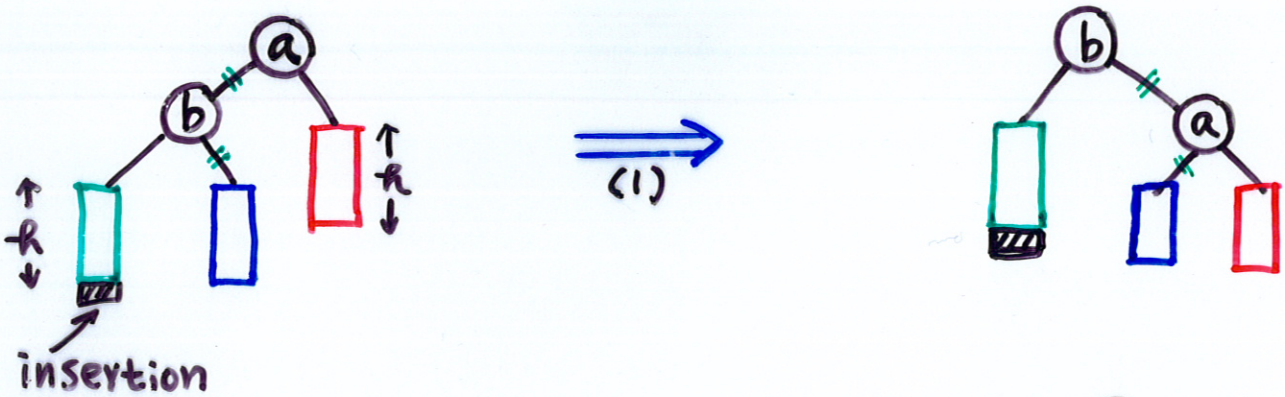
Balanced trees

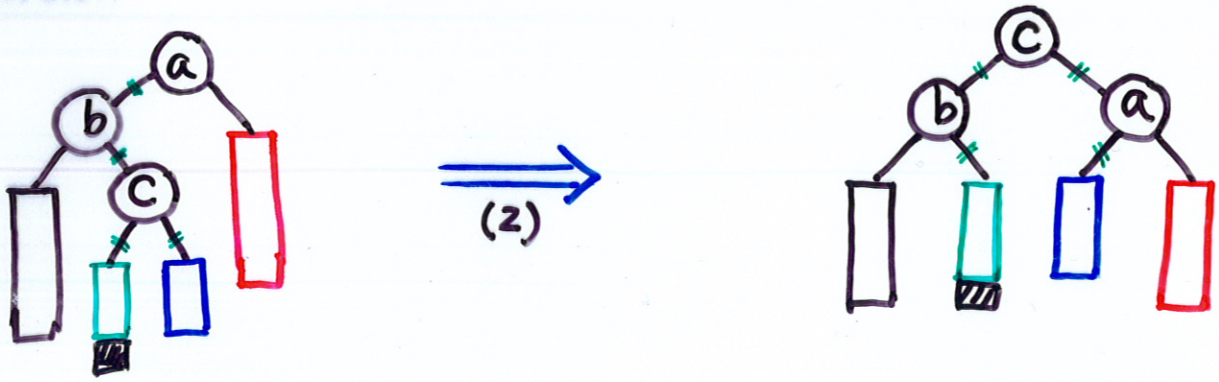
- Wost case : $\log N$
- Weight balanced
- Height balanced: AVL trees, 2-3 trees, 2-3-4 trees

  B-trees

• AVL-trees (Hts difference ≤ 1)





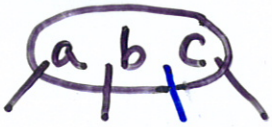


• 2-3-4 trees

2-node

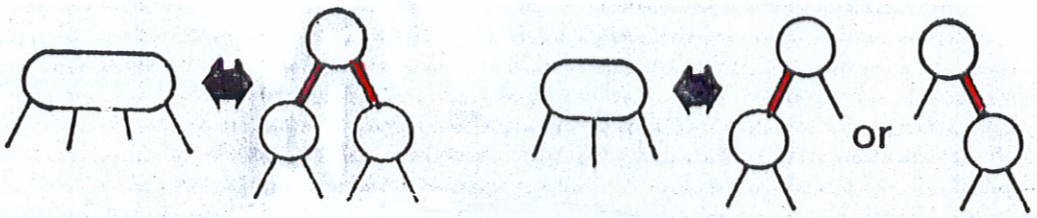
3-node

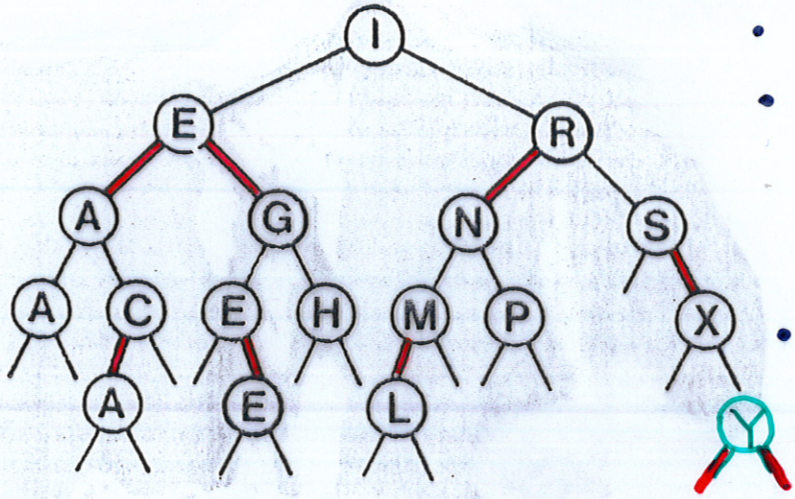
4-node



$b \leq x < c$

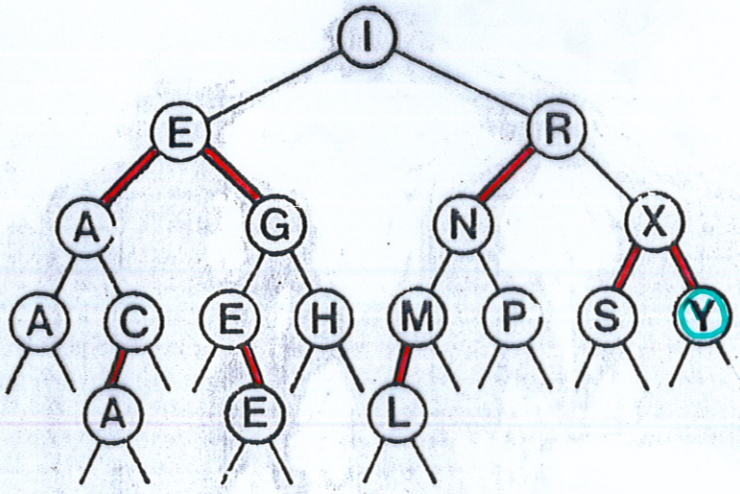
Red-black tree representation (one extra bit)





- No consecutive red links
- Search : as binary trees
  - Simple
  - Save memory
- Insertion ?
  - insert + rotate with 2 red ptrs

Insert Y ⇓

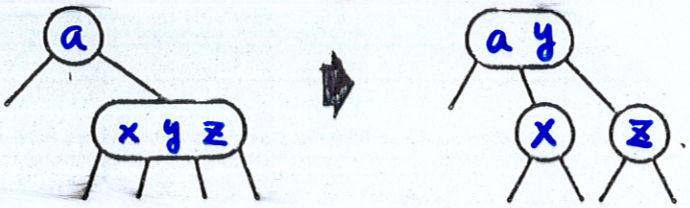


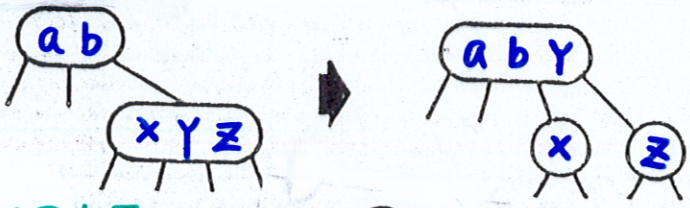
Insertion:

(1) Split 4-node in searching

(2) Insert into bottom node

- $T \le \lg N + 1$





Example: ASEARCHINGEXAMPLE

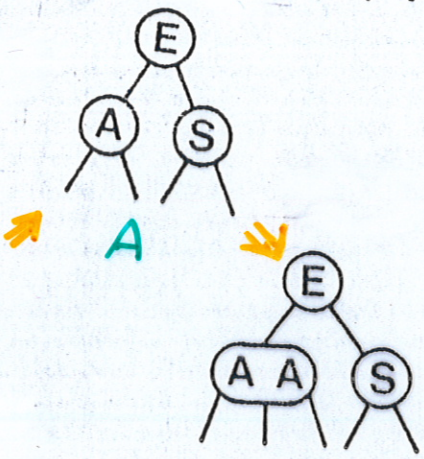
Insert

A

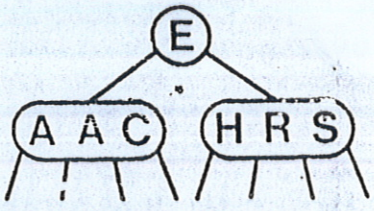
S

E

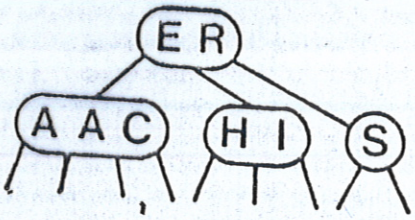
A



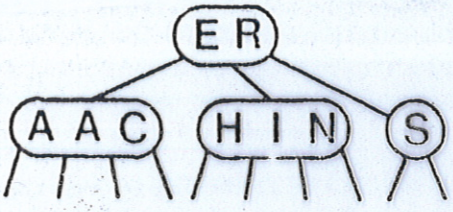
R, C, H



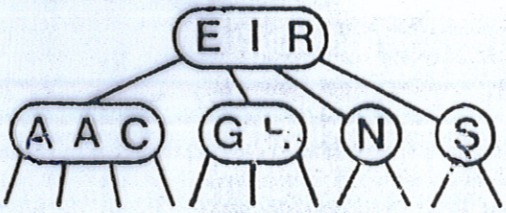
I



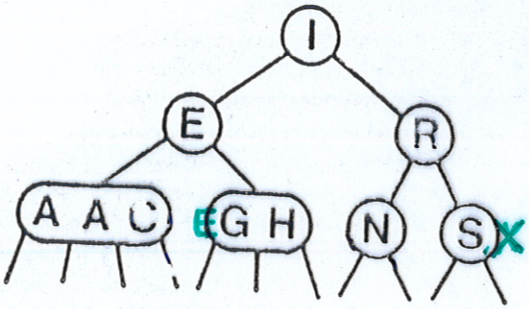
N



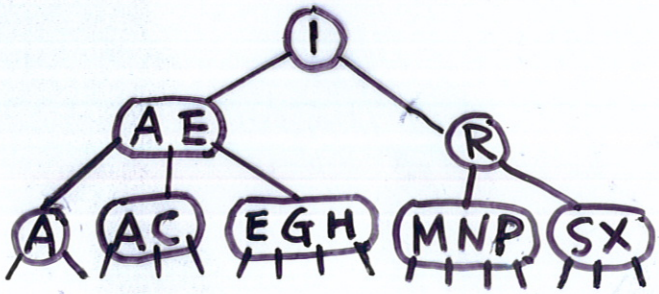
G



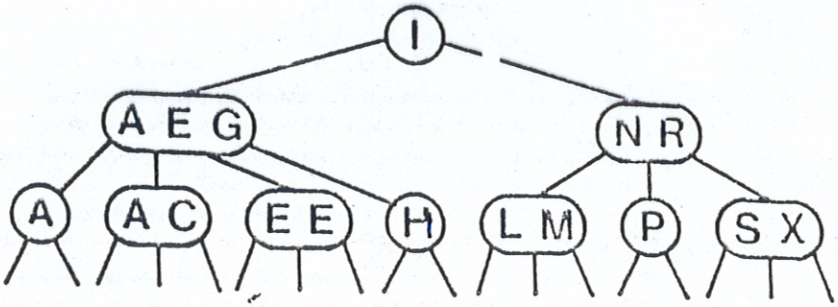
E, X



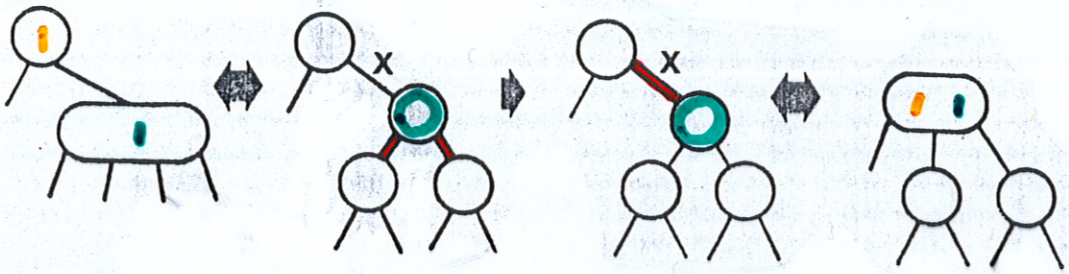
A M P

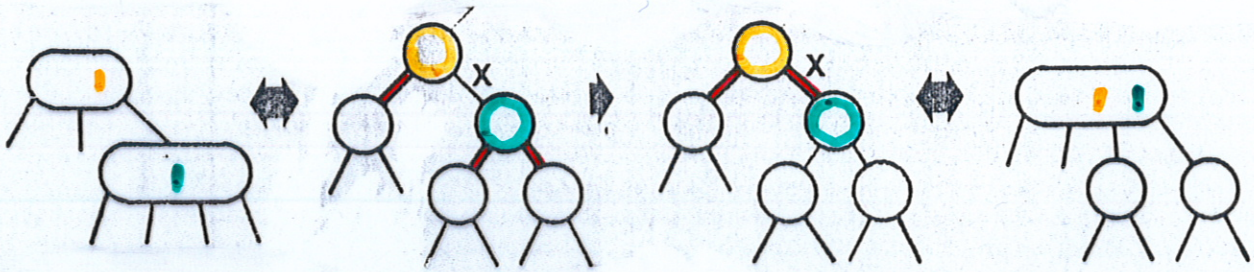


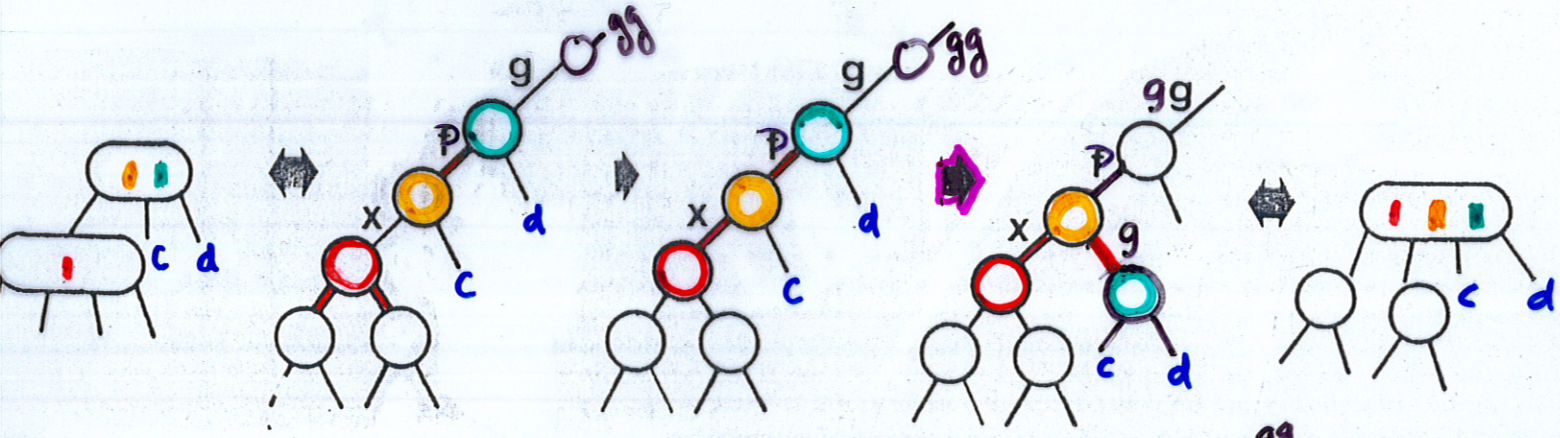
L E

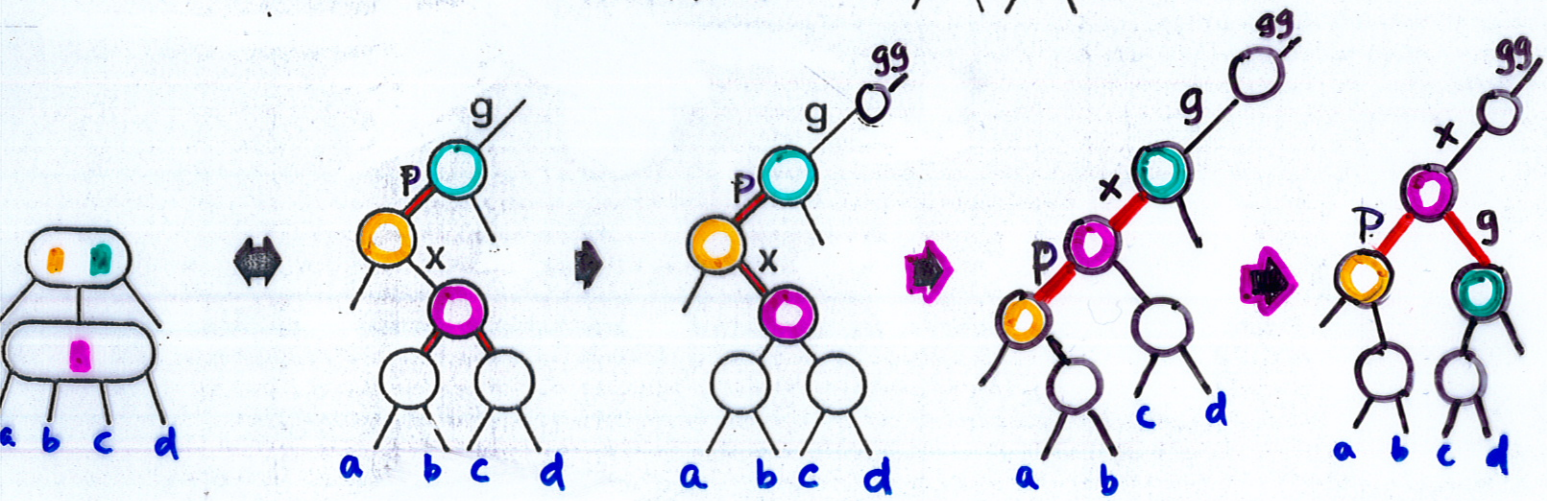


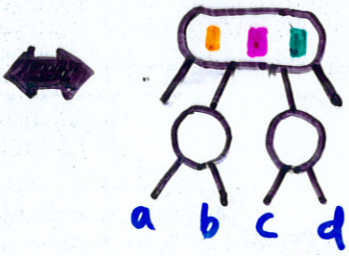
Split


x
x

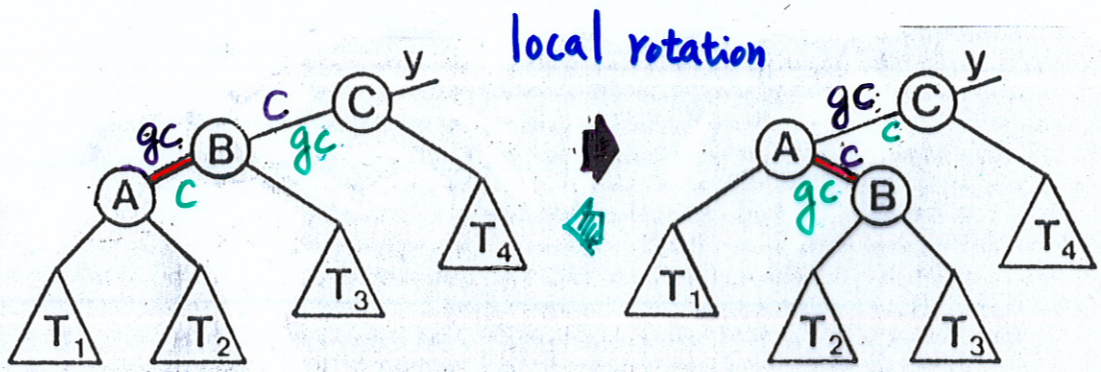

x
x


c d
g gg
p
x
c
d
g gg
p
x
c
d
gg
p
x
g
c d
c d


a b c d
g
p
x
a b c d
gg
g
p
x
a b c d
gg
g
x
p
c d
a b
gg
x
p
g
a b c d

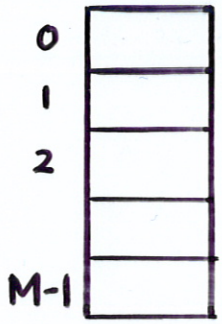

a b c d

Rotate



Hashing

- Scramble Keys
- Compute address in Hash table



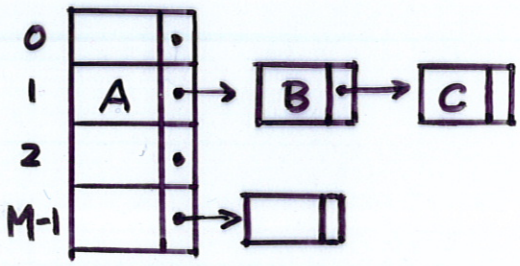
- **Hash Function:**

$$h(\text{Key}) \rightarrow \{0, 1, \cdots, M-1\}$$

  - Similar to random number generator.
  - Example: $h(abc) \rightarrow 1+2+3 \bmod M = 6$

- **Collision**: different keys hash to same address, $x \neq y$, $h(x) = h(y)$

  - **Separate chaining**



  - **Open addressing**

$$K \hookrightarrow \underset{\overset{\parallel}{h(K)}}{h_1(K)}, \; h_2(K), \; \ldots, \; h_M(K)$$

1) **Linear probing**: $h_{i+1}(K) = h_i(K) + 1 \bmod M$

2) **Double Hashing** $h_{i+1}(K) = h_i(K) + u \bmod M$

$$u = g(K)$$

- $u \neq 0$
- $(u, M) = 1$
- $u \neq h(K)$
- $u = M - 2 - K \bmod (M-2)$

- $T = \text{Const}$ (average)
  N (worst)

• Separate chaining : (multi-lists, sequential search)

Key:	A	S	E	A	R	C	H	I	N	G	E	X	A	M	P	L	E
Hash:	1	8	5	1	7	3	8	9	3	7	5	2	1	2	5	1	5

mod 11

0	1	2	3	4	5	6	7	8	9	10
	A	M	C		E		G	H	I	
	A	X	N		E		R	S		
	A				E					
	L				P					

• Linear probing

$h(K),\ h(K)+1,\ h(K)+2, \ldots$

Key:	A	S	E	A	R	C	H	I	N	G	E	X	A	M	P	L	E
Hash:	1	0	5	1	18	3	8	9	14	7	5	5	1	13	16	12	5

mod 19

0	1	2	3	4	5	7	8	9	10	11	12	13	14	15	16	17	18
	A																
S	A																
S	A				E												
S	A	A			E												
S	A	A			E												R
S	A	A	C		E												R
S	A	A	C		E		H										R
S	A	A	C		E		H	I									R
S	A	A	C		E		H	I					N				R
S	A	A	C		E	G	H	I					N				R
S	A	A	C		E	E	G	H	I				N				R
S	A	A	C		E	E	G	H	I	X			N				R
S	A	A	C		E	E	G	H	I	X			N				R
S	A	A	C	A	E	E	G	H	I	X			N				R
S	A	A	C	A	E	E	G	H	I	X		M	N				R
S	A	A	C	A	E	E	G	H	I	X		M	N		P		R
S	A	A	C	A	E	E	G	H	I	X	L	M	N		P		R
S	A	A	C	A	E	E	G	H	I	X	E	L	M	N	P		R

• Clustering.

How about $h(K)+C$? $(C, M) = 1$.

• Double hashing

$h(K),\ h(K)+u,\ h(K)+2u,\ \cdots$

$u = g(K)$

Key	A	S	E	A	R	C	H	I	N	G	E	X	A	M	P	L	E
Hash 1:	1	0	5	1	18	3	8	9	14	7	5	5	1	13	16	12	5
Hash 2:	16	15	12	16	16	14	9	8	3	10	12	10	16	4	1	5	12

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	$\boxed{A}$																	
$\boxed{S}$	A																	
S	A				$\boxed{E}$													
S	$\boxed{A}$				E												$\boxed{A}$	
S	A				E												A	$\boxed{R}$
S	A		$\boxed{C}$		E												A	R
S	A		C		E			$\boxed{H}$									A	R
S	A		C		E			H	$\boxed{I}$								A	R
S	A		C		E			H	I					$\boxed{N}$			A	R
S	A		C		E		$\boxed{G}$	H	I					N			A	R
S	A		C		$\boxed{E}$		G	H	I	$\boxed{E}$				N			$\boxed{A}$	R
S	A		C		$\boxed{E}$		G	H	I	E				N	$\boxed{X}$		A	R
S	$\boxed{A}$		C		E		G	H	I	E	$\boxed{A}$			$\boxed{N}$	X		$\boxed{A}$	R
S	A		C		E		G	H	I	E	A		$\boxed{M}$	N	X		A	R
S	A		C		E		G	H	I	E	A		M	N	X	$\boxed{P}$	A	R
S	A		C		E		G	H	I	E	A	$\boxed{L}$	M	N	X	P	A	R
S	$\boxed{A}$		$\boxed{C}$		$\boxed{E}$	$\boxed{E}$	G	$\boxed{H}$	I	$\boxed{E}$	A	L	$\boxed{M}$	N	$\boxed{X}$	P	$\boxed{A}$	R

Performance

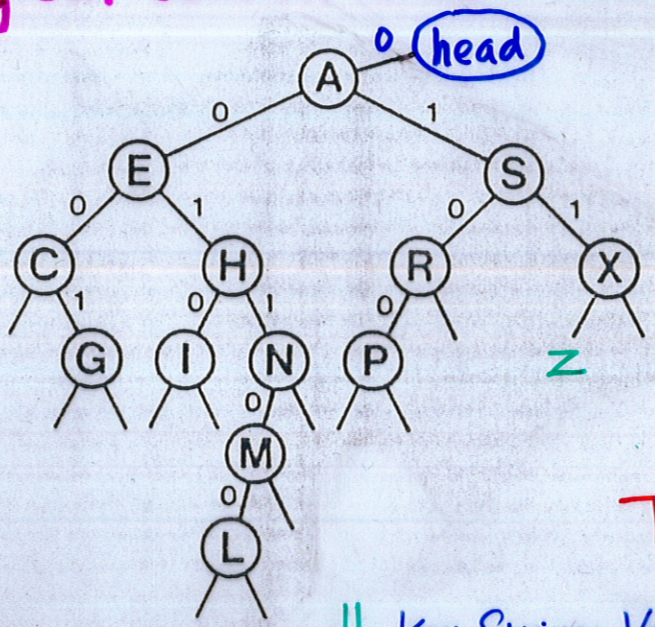
	Unsuccessful	Successful
Separate chaining	$1+\frac{\alpha}{2}$	$\frac{\alpha+1}{2}$
Linear probing	$\frac{1}{2}+\frac{1}{2(1-\alpha)^2}$	$\frac{1}{2}+\frac{1}{2(1-\alpha)}$
Double Hashing	$\frac{1}{1-\alpha}$	$-\frac{1}{\alpha}\ln(1-\alpha)$

$\alpha = \frac{N}{M}$

load factor

Radix Searching

Digital search tree : branch by bits, not key comparisons.



A	00001
S	10011
E	00101
R	10010
C	00C11
H	01000
I	01001
N	01110
G	00111
X	11000
M	01101
P	10000
L	01100
Z	11010

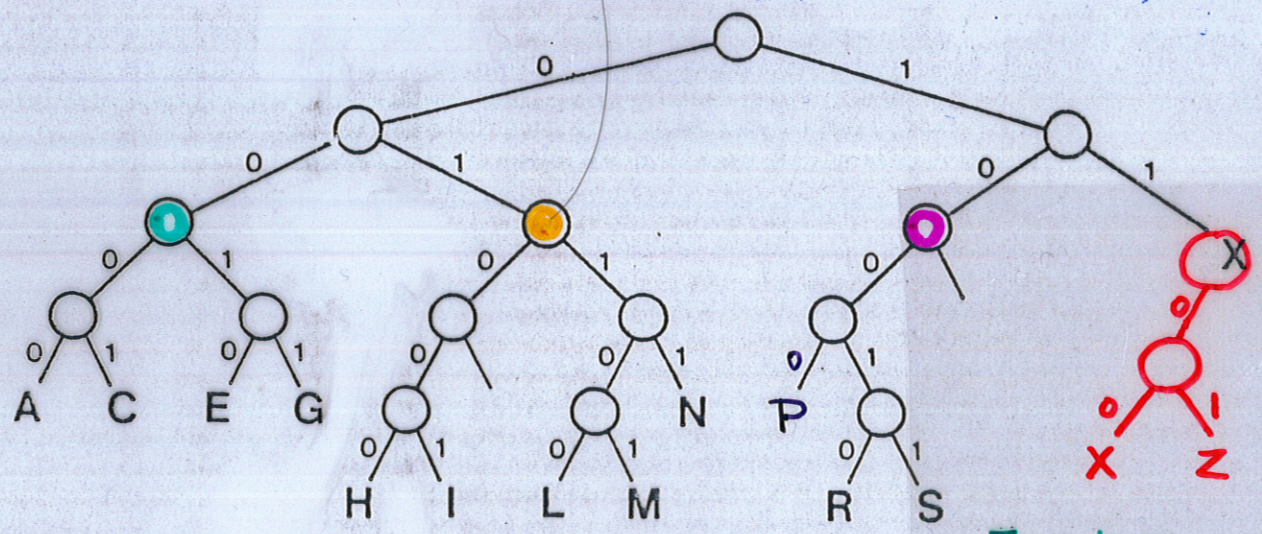
Search M = 01101

Insert Z = 11010

$$T = \begin{cases} \lg N & \text{(average)} \\ b & \text{(worst)} \end{cases}$$

⇓ Key Strings Very Long

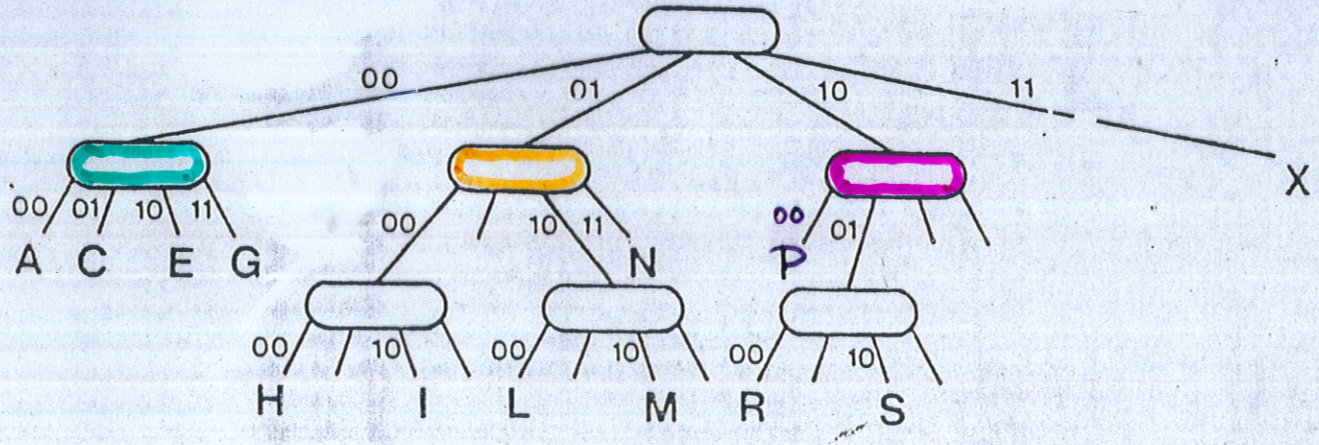
Radix search tree (trie) : compare Keys at leaf node only



Insert Z = 11010

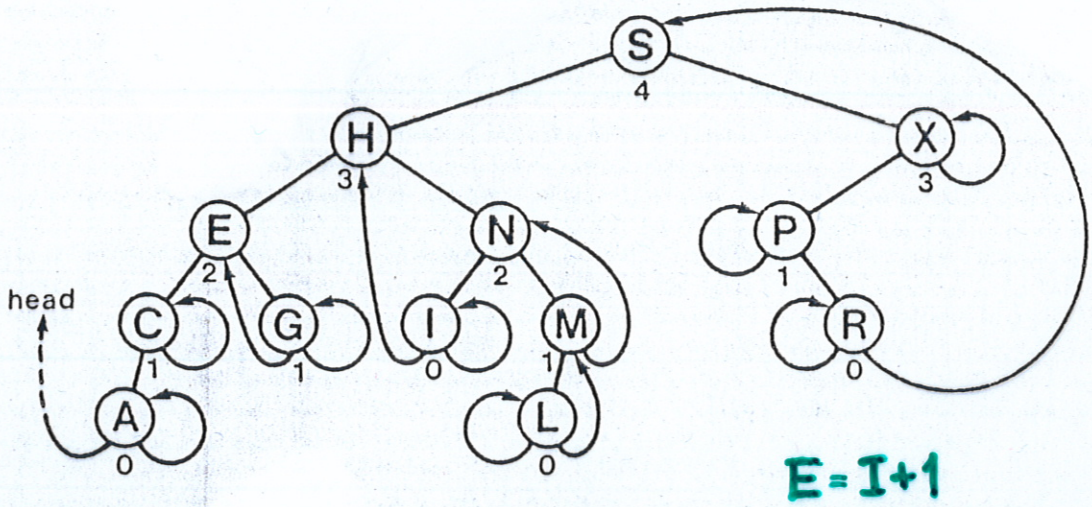
- Internal ↔ External node
- 1-way branching

Multiway Radix searching

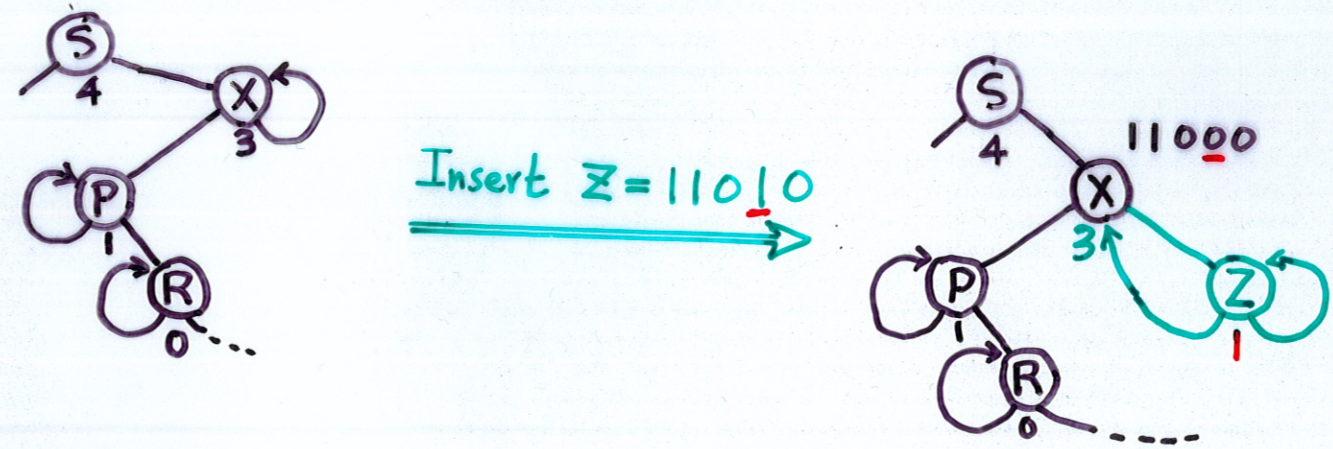


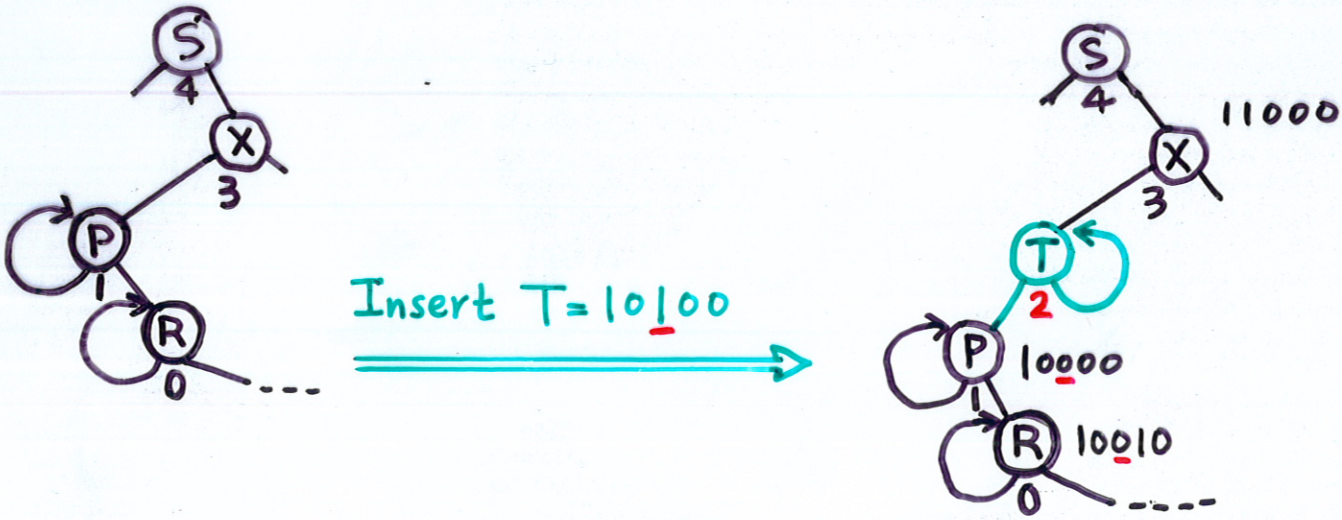
Patricia (Practical Algorithm to Retrieve Information Coded in Alphanumeric)

- index → bit examined
- upward link → external node



Insertion:





External Searching

- disk is made of Pages = Continuous blocks
- Page transfer dominates searching time

• Indexed Sequential Search (ISAM)

Internal memory

$\boxed{* \ 1 \ e \ 2 \ n \ 3 \ x}$ master index

Sequential Search

Disk 1:	* 1 c 2 e	A A A C	E E E E
Disk 2:	e 1 i 2 n	E G H I	L L M N
Disk 3:	n 1 r 2 x	N P R R	S T X X
	page 0	page 1	page 2

T = 2 disk accesses.

Disadvantages : Insert B ?

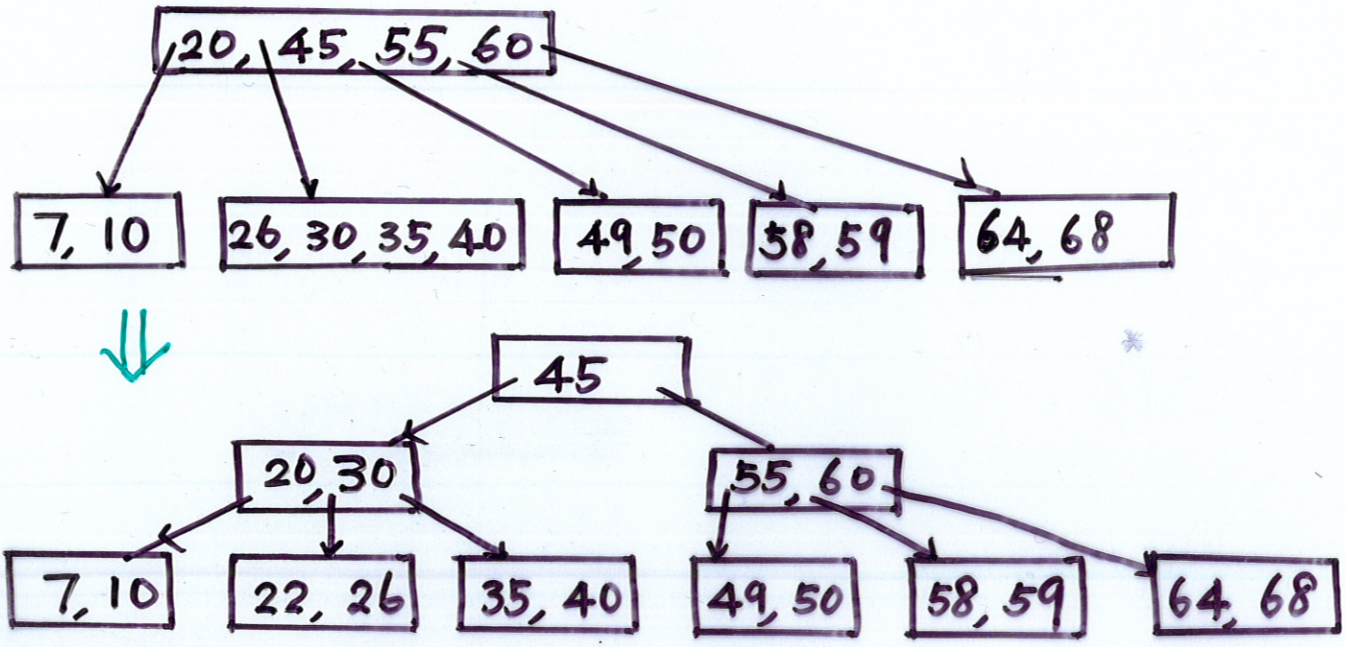
B-tree

$n \leq$ # records in page $\leq 2n$ (except root)

$n=2$

(1) Insert 22

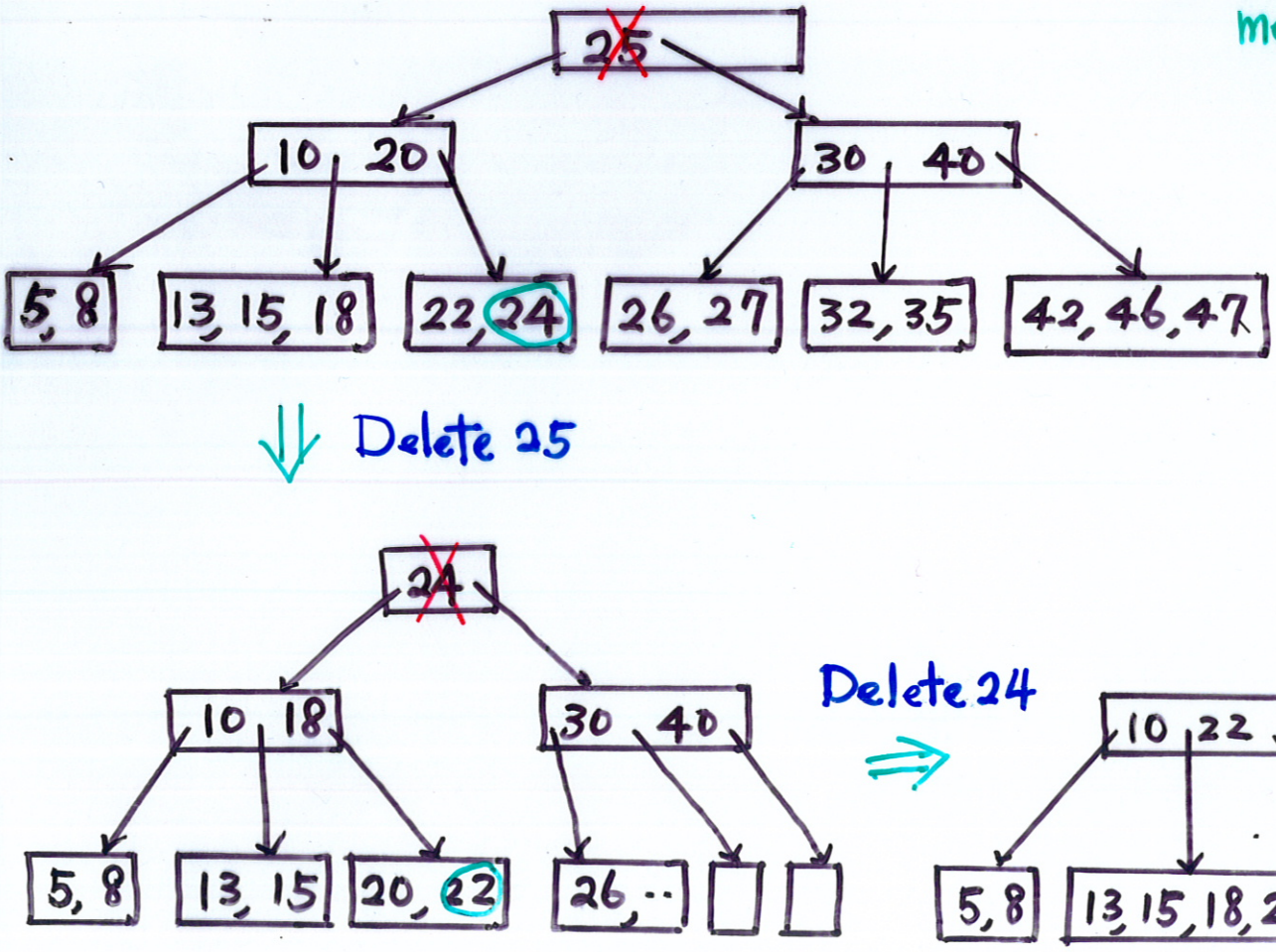
• # records $= 2n \Rightarrow$ split



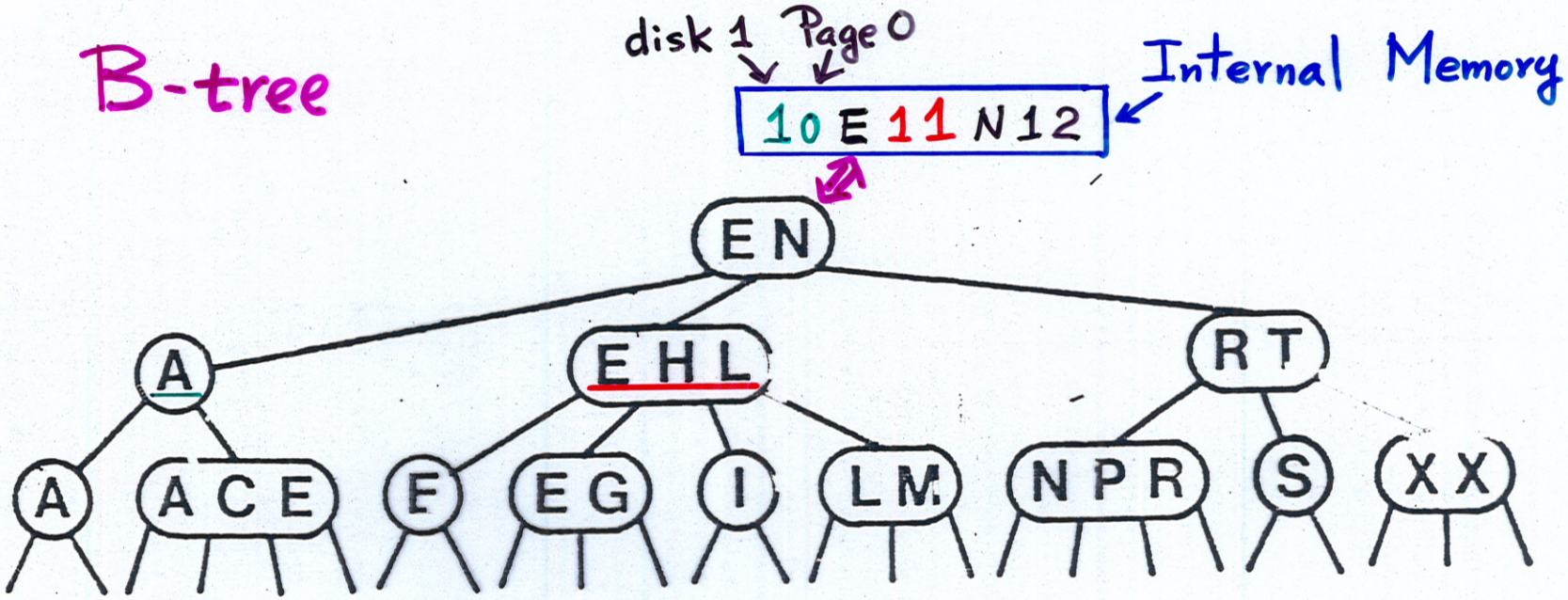
(2) Delete 25, 24

• # records $= n$

$\Rightarrow$ balance or merge



B-tree



Sequential Search

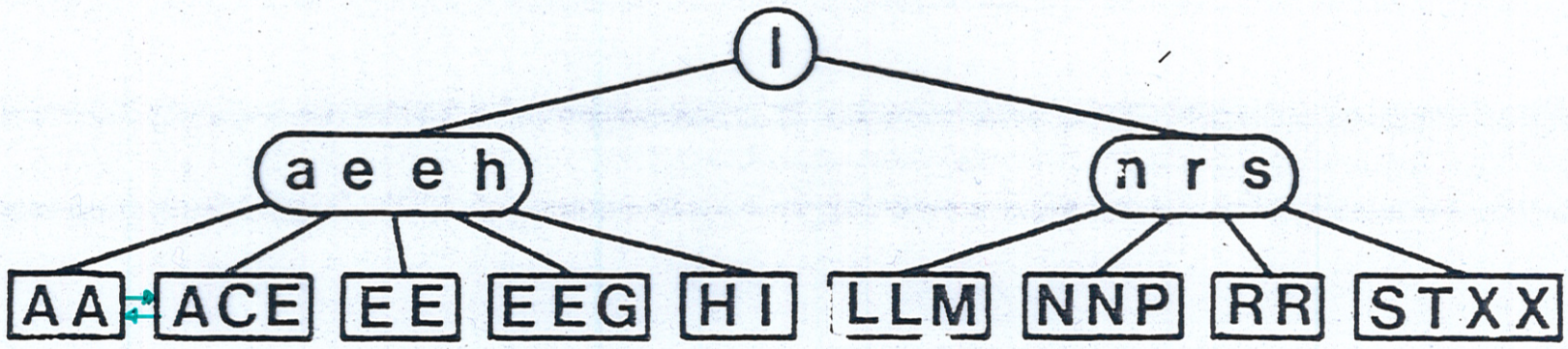
Disk 1:	20 A 21	22 E 30 H 31 L 32	40 R 41 T 42
Disk 2:	0 A 0	0 A 0 C 0 E 0	0 E 0
Disk 3:	0 E 0 G 0	0 I 0	0 L 0 M 0
Disk 4:	0 N 0 P 0 R 0	0 S 0	0 X 0 X 0

M_I : # Keys in Internal node : Both NOT TOO Large

M_B : " External node Small !!!

B^+-tree : Increase M_I,

10 Internal Memory



← Keys in sorted order

Disk 1:	11	l	12			20	a	21	e	22	e	30	h	31	32	n	40	r	41	s	42
Disk 2:	A		A			A		C				E			E		E				
Disk 3:	E		E		G	H		I							L		L		M		
Disk 4:	N		N		P	R		R							S		T		X		X

- Supports previous(K) operations
  Next(K)

Extendible Hashing

Page Capacity = 4

Insert: E, X, T, E

00101 E
00101 E
10100 T
11000 X

Disk 1: 20
Disk 2: E E T X

R = 10010

0: 00 01 E
00101 E
1: 10010 R
10100 T
11000 X

Disk 1: 20 21
Disk 2: E E R T X

N = 01110
A = 00001

0: 00001 A
00101 E
00101 E
01110 N
1: 10010 R
10100 T
11000 X

Disk 1: 20 21
Disk 2: A E E N R T X

L = 01100

00: 00001 A
00101 E
00101 E
01: 01100 L
01110 N
10: 10010 R
11: 10100 T
11000 X

00 01 10 11

Disk 1: 20 21 22 22
Disk 2: A E E L N RTX

Internal Memory

$\boxed{10\ 11}$

↳ bit = '0' ↳ bit = '1'

S = 10011

E = 00101

A = 00001

000:	00001	A
	00001	A
001:	00101	E
	00101	E
	00101	E
010:	01100	L
011:	01110	N
100:	10010	R
101:	10011	S
110:	10100	T
111:	10101	X

010 011

Disk 1: 20 21 22 22 30 30 30 30

Disk 2: A A E E E L N

Disk 3: R S T X

100 101 110 111

30 30 31 31

R S T X Z

- Insert Z = 11010 ?
- 2 disk accesses (like Indexed Sequential)

  Efficient insertion (like B-tree)
- Hash (Key) → random bit string