Bubble sort

Best case:

NO swaps, only comparison.

$$\sum_{i=1}^{n} \sum_{j=1}^{n-i} 1$$

$$= \sum_{i=1}^{n} (n-i-1)+1 \quad [\text{upper} - \text{lower} + 1]$$

$$= \sum_{i=1}^{n} n-i$$

$$= (n-1) + (n-2) + \cdots 2 + 1$$

$$= \frac{n(n-1)}{2}$$

$\therefore \Omega(n^2)$

Worst case:

Swaps + Comparison.

$$\sum_{i=1}^{n} \sum_{j=1}^{n-i} (1+1)$$

$$= 2 \cdot \frac{n(n-1)}{2}$$

$$= n(n-1)$$

$\therefore O(n^2)$

Avg case:

comparison + (swap, no swap)
(1 , 0)

$$E[x] = (0+1) \times \frac{1}{2}$$

$P(\text{swap}) = P(\text{no swap}) = 1/2$

$$= 1/2$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n-i} \{E[x] + 1\} \rightarrow \text{comparison}$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n-i} (1/2 + 1) = 3/2$$

$$= 3/2 \; \frac{n(n-1)}{2}$$

$$= \frac{3}{4} n(n-1)$$

$\therefore \Theta(n^2)$

In place algorithm. hence space complexity is $O(1)$.

Insertion Sort

Best case: no swap, only comparison

for every i, inner loop will execute 1 time. outside loop will execute $(n-1)$ times. hence, we will do $(n-1)$ comparisons.

$\therefore \Omega(n)$

Worst case: 5 4 3 2 1

for 2nd item we need 1 comp & 1 swap.

" 3rd " " " " 2 " & 2 "

" 4th " " " " 3 " & 3 "

⋮

" nth " " " " $(n-1)$ " & $(n-1)$ " .

$$\therefore \text{total} = \frac{n(n-1)}{2} + \frac{n(n-1)}{2}$$

$$= O(n^2)$$

Avg case: $\Theta(n^2)$

Inplace algorithm, space $O(1)$

4 { 2 5 3 }

Selection sort

Best case:

Swaps → $\sum_{i=1}^{n-1} 1 = n-1-1+1 = (n-1)$

Comparison → $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1} (n-i-1+1)$

$= \sum_{i=1}^{n-1} (n-i)$

$= \frac{n(n-1)}{2}$

$\therefore (n-1) + \frac{n(n-1)}{2} = \Omega(n^2)$

Worst case:

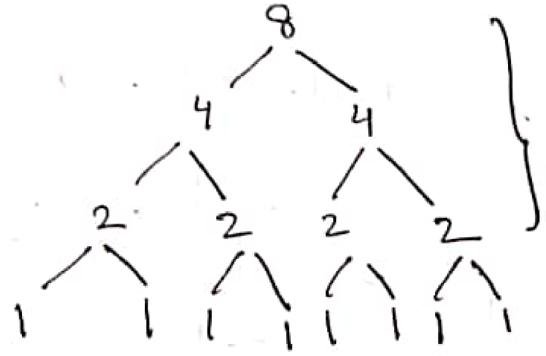
swaps $= n-1$

comparison $= \frac{n(n-1)}{2}$

$\therefore O(n^2)$

In Place, $O(1)$

Merge sort



height = 3 $\quad n = 8$

$= \log_2(8)$

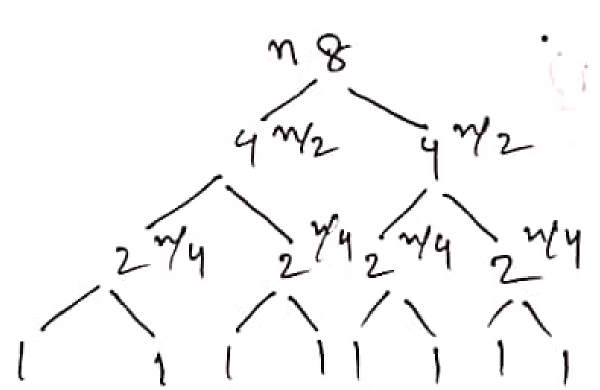
$= \log_2(n)$

at each level we merge the arrays.

we need $(n-1)$ comparison to merge 2 arrays.

to merge 2 arrays of length 4 we need $(2*4-1)$

= 7 comparison.



$1(n-1) = 2^0\left(\frac{n}{2^0}-1\right)$

$2(n/2-1) = 2^1\left(n/2^1-1\right)$

$4(n/4-1) = 2^2\left(n/2^2-1\right)$

$8(n/8-1) = 2^3\left(\frac{n}{2^3}-1\right)$

$2^i\left(\frac{n}{2^i}-1\right)$ $\quad$ i = level

$$\text{total work} = \sum_{i=0}^{\log(n)-1} 2^i\left(\frac{n}{2^i}-1\right)$$

[as $\log 8 - 1 = 2$ & we have 3 level $[0,1,2]$

$$= \sum_{i=0}^{\log(n)-1} (n-2^i)$$

$$= \sum_{i=0}^{\log(n)-1} n - \sum_{i=0}^{\log(n)-1} 2^i$$

$$= (\log(n) - 1 - 0 + 1)\, n - \sum_{i=0}^{\log(n)-1} 2^i$$

$$= n\log n - (1 + 2 + 4 + \cdots + 2^{\log(n)-1})$$

$$= n\log n - \left(\frac{2^{\log n - 1 + 1} - 1}{2-1}\right)$$

$$= n\log n - (2^{\log n} - 1)$$

$$= n\log n - n + 1 \quad [x^{\log_x n} = n]$$

$$= O(n\log n)$$

if $|x| > 1$

$$\sum = \frac{x^{K+1} - 1}{x - 1}$$

here,

$x = 2$

$K = \log(n) - 1$

$n = 8$,

total comparison $= 8\log 8 - 8 + 1$

$= 17$

$$T(n) = \underbrace{T(n/2)}_{\text{split left}} + \underbrace{T(n/2)}_{\text{split right}} + \underbrace{(n-1)}_{\text{comparison at each level}}$$

$$\Rightarrow T(n) = 2T(n/2) + (n-1)$$

$\therefore T(8) = 2T(4) + 7$

$= 2[2T(2) + 3] + 7$

$= 2[2[2T(1) + 1] + 3] + 7$

$= 2[2(2.0 + 1) + 3] + 7$ [$T(1) = 0$, we don't compare when we have 1 element]

$= 2[2 + 3] + 7$

$= 10 + 7$

$= 17$ comparisons.

Quick sort

Best case: Pivot divides the array in 2 equal sub array.

n/2	P	n/2

$T(n) = 2T(n/2) + (n-1)$

→ same as merge sort
$\Omega(n \log n)$

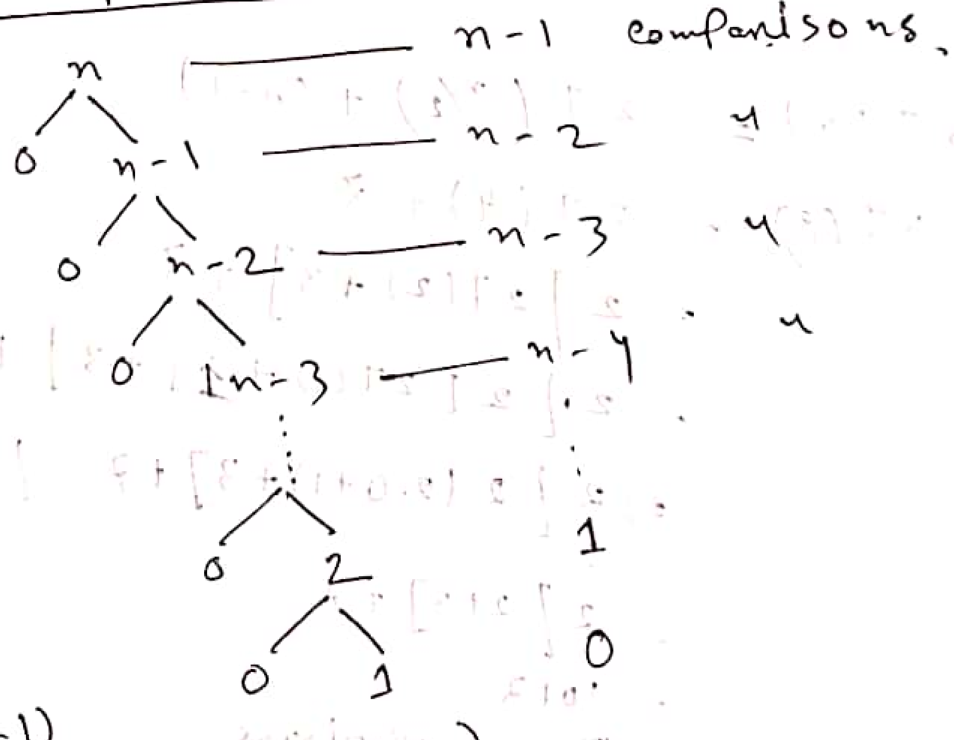
Worst case: when array is sorted, reverse sorted or all the elements are same.

when we pick the largest / smallest element as pivot, this pivot will divide the array in (0, n-1).

P	n-1

$T(n) = T(0) + T(n-1) + (n-1)$

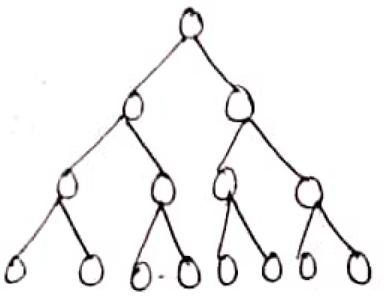
$= T(n-1) + (n-1)$



1 to n-1

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$$

Heap sort



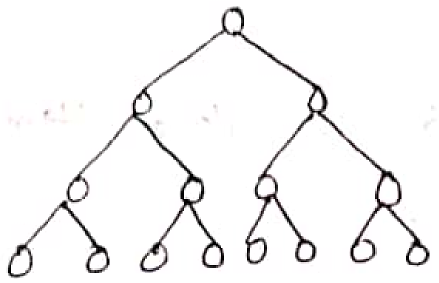
Build-heap

~~mone~~ Half of total nodes ($n/2$) are in the last level. hence, to build the heap it is sufficient to start from the previous level of the last level. That means from $n/2$ to 1.

For each of the $n/2$ nodes we need to heapify that node. At most a node could go downwards $\log n$ time which is the height of the tree.

hence, we can say time complexity for Build heap is $\rightarrow O(n/2 \log n) \cong O(n \log n)$

So But we could reduce it to $O(n)$ if we do exact calculation, as it depends on which level the node is, how much it would go deeper.



1

2

4

8

$n = 15 \cong 16$

level	No. of nodes	No. of level it could go down at worst	No. of comparison at each level	Total
0	1	3	2	1*6
1	2	2	2	2*4
2	4	1	2	4*2
3	8	0	2	8*0

$$\text{Total} = \underbrace{\left(0 * \frac{n}{2}\right)}_{\text{nodes in the last level (3)}} + \underbrace{\left(2 * \frac{n}{4}\right)}_{\text{nodes in level 2}} + \underbrace{\left(4 * \frac{n}{8}\right)}_{\text{nodes in level 1}} + \underbrace{\left(6 * \frac{n}{16}\right)}_{\text{nodes in level 0}} + \cdots$$

$$= n \sum_{i=0}^{\lg n - 1} \frac{2i}{2^{i+1}}$$

$$= n \sum_{i=0}^{\lg n - 1} \frac{i}{2^i}$$

$$= n \left[\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots\right]$$

$$= 2n = O(n)$$

2n comparisons at worst case.

$$S = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots + 0 \quad \text{— (i)}$$

$$S/2 = \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \frac{4}{32} + \cdots + 0 \quad \text{— (ii)}$$

$$\text{(i)} - \text{(ii)} \Rightarrow S/2 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8}$$

Extract root :

we do $(n-1)$ iterations to place items from the end to 2nd position.

1	2	- - -	n

$$\underbrace{2 \; \cdots \; n}_{n-2+1 = n-1}$$

at each iteration we extract root, replace it with the last element and heapify the tree.

at worst we would need to traverse from root to all the level down. which is $\log n$. But at each iteration the heap size reduces :

$$\sum_{i=2}^{n} 2 \log(i)$$

→ no. of comparison at each level.

→ not $\log n$ as, we won't go down $\log n$ level each time because the heap size decreases.

→ counting from the last level.

$$\sum_{i=2}^{n} 2 \log(i)$$

$= 2n\log n - 2n\log(e) + \log(2\pi) + \log n$

$= O(n\log n)$

Total complexity $= O(\text{Build heap}) + O(\text{extract root})$

$= O(n) \text{ or } O(n\log n) + O(n\log n)$

$= O(n\log n)$