Selection Short

Best → O(1)

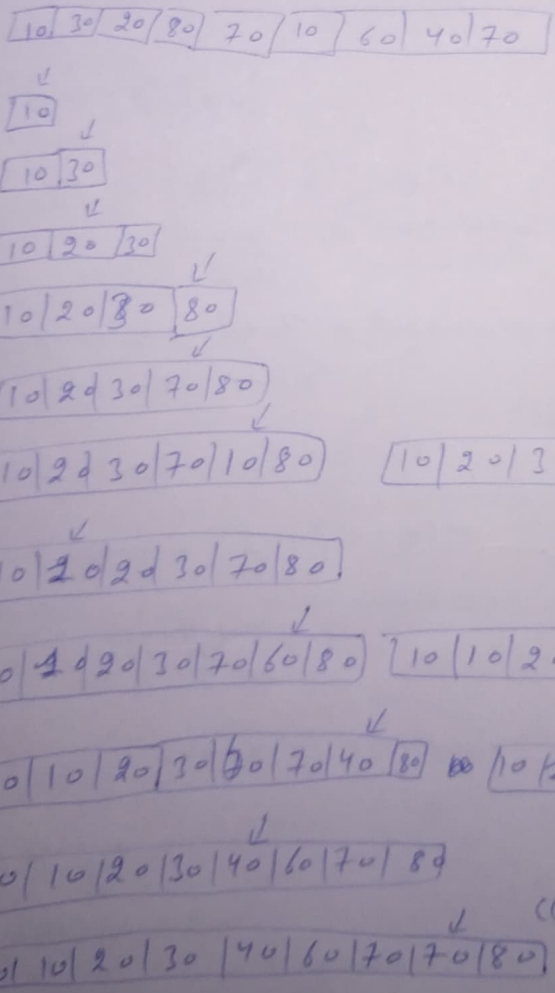
① take one element and compare it with every element

Bubble Short

① make a bubble of 2 element and compare them
② from this max element is goes to ending index
③ then again we make a bubble of 2 elemnt and compare them with n-2-i index

Insertion Short

① ~~make~~ firstly make first element as shorted (Assume that is diff array)
② the go the next position and check above array for that position
③ every time we have a array for check that is already shorted



J := 0 to a[j] ; item

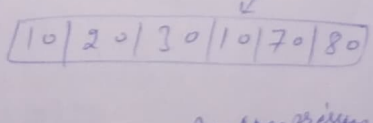
① shorted → comparison → n-1
swap → 0
O(n)

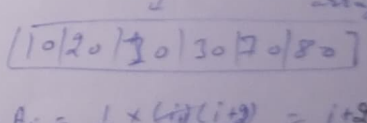
② Input is shorted in reverse
comparison = $\frac{n(n-1)}{2}$
swap = $\frac{n(n-1)}{2}$
$O(n^2)$

③ Random = $A_{i+1} = \sum_{j=1}^{i} j \times P_j$

J = key go to it location
P_j = Probability of one element in array

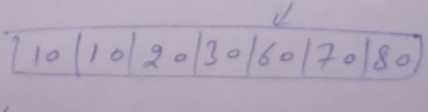
$i \le j \le i+1$





Avg comparison → $A_{i+1} = \frac{1}{i+1} \times \frac{(i+1)(i+2)}{2} = \frac{i+2}{2}$

$= \frac{1}{i+1} \sum_{j=1}^{i+1} j =$



Total no of comparisons =

$C(n) = \sum_{i=1}^{n-1} A_{i+1} = \frac{i+2}{2} \Rightarrow \sum_{i=0}^{n-1} \frac{i}{2} + \sum_{i=0}^{n-1} 1$

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

movement : $M_i = \frac{i+(i-1)+(i-2)+\cdots-2+1}{2i} = \frac{i+1}{2}$

Total no of movement = $\sum_{i=1}^{n-1} M_i = \frac{1}{2}\cdot\frac{n(n-1)}{2} + \frac{(n-1)}{2} \Rightarrow \frac{(n-1)(n+2)}{4}$

Average case = $O(n^2)$

for time complexity = comparison + movement

Selection short:- Best case → Sharted → comparison → $C(n) = \sum_{i=1}^{n-1}(n-i)$

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

Movement = 0

Reverse unsharted → comparison = $C(n) = \sum_{i=1}^{n-1}(n-i)$

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

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

↓

swaping cost temp = a[i]
a[i] = a[j]
a[j] = temp

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

Movement = Total no of swap operations
= $(1-P_i) \times (n-1)$

$P_1 = P_2 = P_3 \cdots P_n = \frac{1}{n}$

$M(n) = \left(1-\frac{1}{n}\right) \times (n-1)$

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

Bubble Sort :- sorted order → Best case = $C(n) = \sum_{i=1}^{n-1}(n-i)$

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

OR (On)

movement = 0

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

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

Random order $= c(n) = \sum_{j=1}^{n-i+1} (\overline{n-i+1} - j) \cdot p_j$

$1 \le j \le n-i+1$

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

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

$$= \frac{2n-2i+2 - n-i+2}{2} = \frac{n-i}{2}$$

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

Quick Sort

Space of stack $= \log(n)$

acending order = (omparison $= C(n) = n-1 + C(n-1) + \cdots$
(worst (ase) $C(n) = (n-1) + (n-2) + (n-3) + \cdots + 1$

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

Movemen = 0

Reverse order = Comparison $= C(n) = (n-1) + (n-2) + (n-3) \cdots + 1$

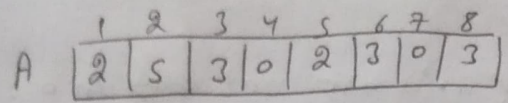
$= \frac{n(n-1)}{2}$

Movement $= \begin{cases} \frac{n-1}{2} & \text{if } n \text{ is odd} \\ \frac{n}{2} & \text{if } n \text{ even} \end{cases}$

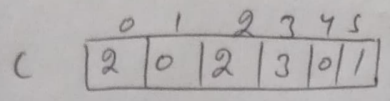
Random order

merge, Radix

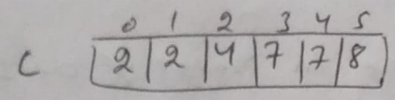
Counting Sort



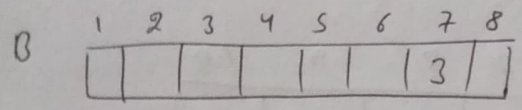
start from last and update C array every time

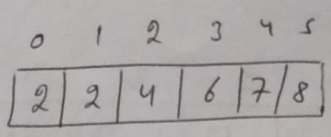


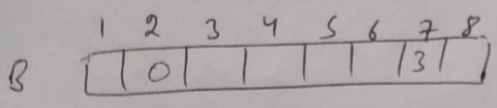
frequency

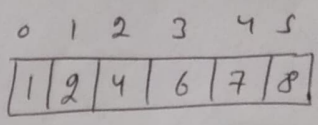


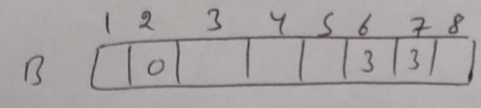
cumulative sum

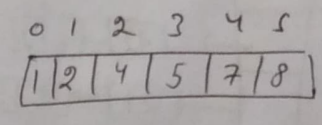


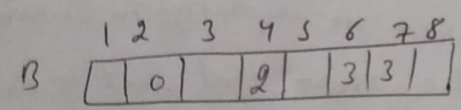


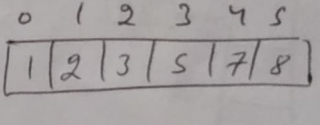


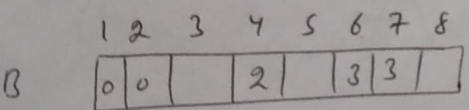


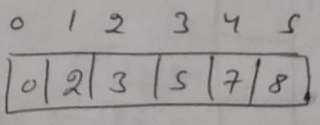


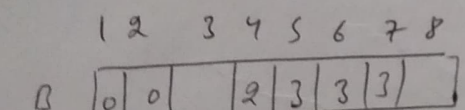


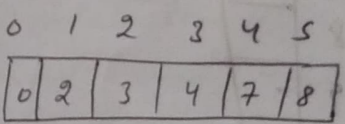


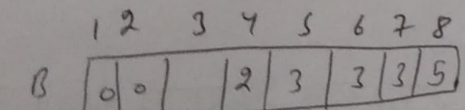


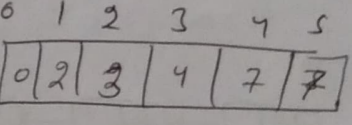


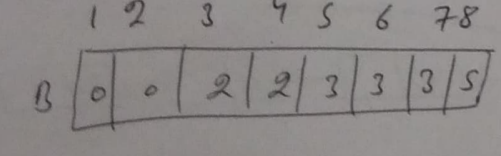


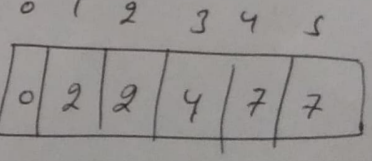












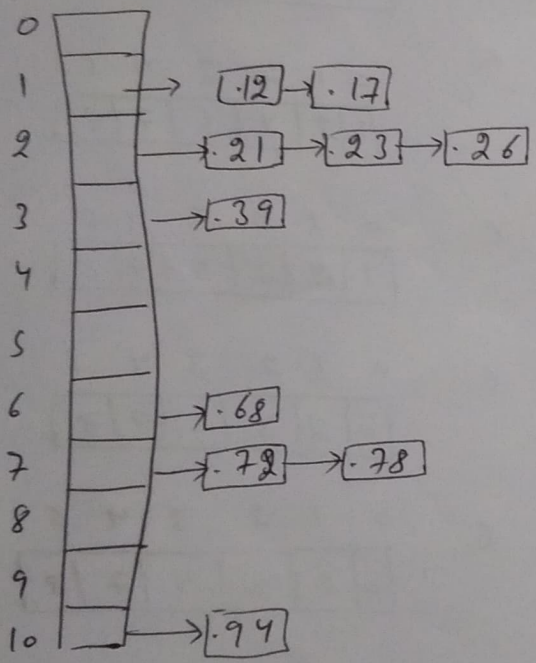
$O(n)$

Bucket Sort

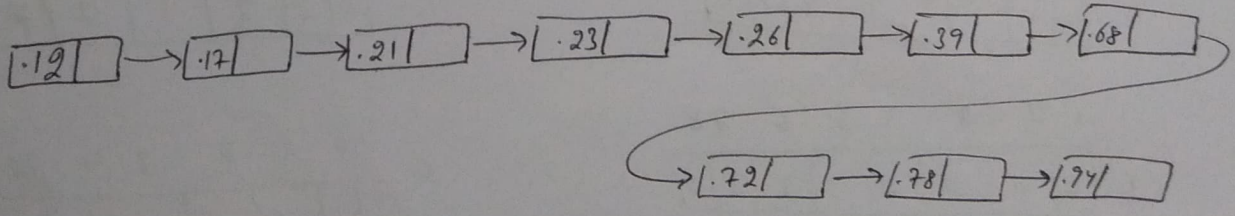
$$\text{Floor}\left(\frac{n*(\text{element}-\text{min})}{1+\text{max}-\text{min}}\right)$$ → if no's are not only in o'1 range

Floor (n * A[i]) → if no's are in range of 0 & 1

	1	2	3	4	5	6	7	8	9	10
A	.78	.17	.39	.26	.72	.94	.21	.12	.23	.68



when no is placing in Bucket also short that list of Buckets this time



Heap Sort

Heapify

Build Heap

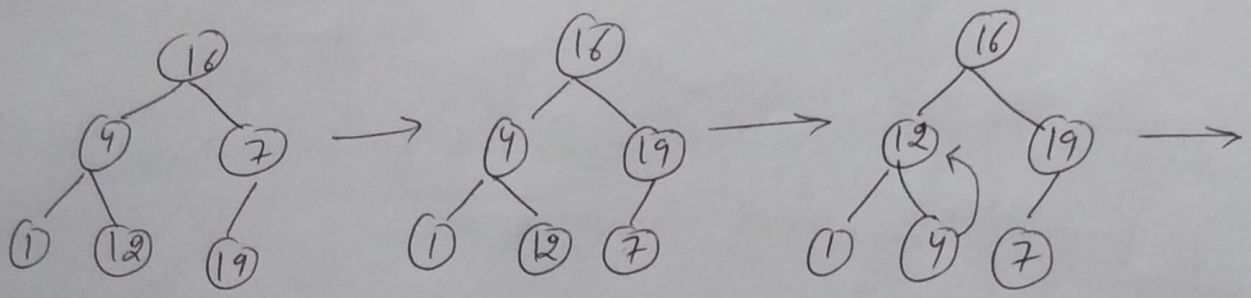
Heap Sort

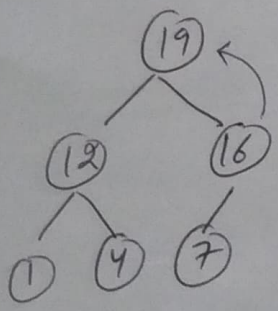
$n \log n$

16	4	7	1	12	19

for ~~ma~~ increasing order → make max heap

" decreasing order → make min heap





here we ~~make~~ max heap now we ~~delete~~ element from root and replace it with lowest element (leaf node) element and delete that leaf element and again call heapify function

Simultaionsly repeat this process