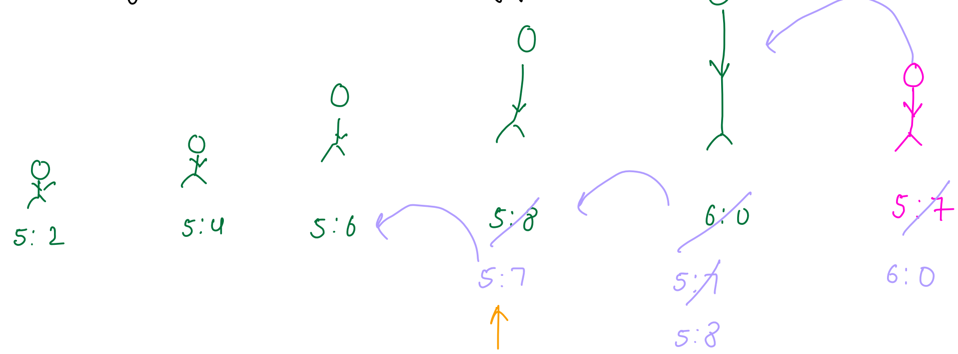
Lecture : Insertion and Quick sort

Agenda

- Insertion sort
- Rearrange the array
- Quick sort.

Standing in a line. [Analogy]



Correct position.

Insertion sort

sorted | unsorted

Ex: arr[5] = { 2 | 1 3 -2 0 }

sorted   unsorted

1   2 | 3   -2   0

-2   ~~1~~   ~~2~~   ~~3~~ | ~~-2~~ 3   0

~~-2~~ -1   ~~-2~~ 2

-2   ~~1~~ 0   ~~0~~ 1   ~~2~~   ~~0~~ 2   ~~3~~ | ~~0~~ 3

-2   0   1   2   3

Worst case:

arr[] = [5 4 3 2 1]

4 5 3 2 1

4 3 5 2 1

3 4 5 2 1

3 4 2 5 1

3 2 4 5 1

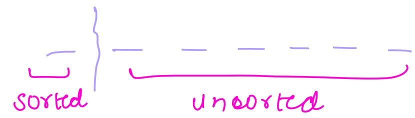
2 3 4 5 1

2 3 4 1 5

2 3 1 4 5

2 1 3 4 5

1 2 3 4 5




```
void insertionSort(int[] arr) {

    for (i=1; i< arr.length; i++) {
        int j = i-1;
        while (j>=0 && arr[j] > arr[j+1]) {
            swap(arr, j, j+1);
            j--;
        }
    }
}
```

TC: $O(n^2)$

SC: $O(1)$

Inplace sorting

Concept [Nuts and bolts problem]

* Given n nuts and n bolts of different sizes.
  There is 1:1 mapping

* Match nuts with bolts

* Can't compare nuts with nuts and bolts with bolts.

nuts: [2 1 4 3 5]

bolts: [4 2 1 5 3]

Brute force: Sort both arrays.

nuts: { 1 2 3 4 5 }

bolts: { 1 2 3 4 5 }

$\Downarrow$

Can't do that as can't compare nuts with nuts & bolts with bolts.

Partition concept:

n1	n2	n3
n4	n5	n6

b1	b2	b3
b4	b5	b6

$\underbrace{n1 \quad n2 \quad n4}_{< b1}$ b1 n5 $\underbrace{n3 \quad n6}_{> b1}$

$\underbrace{b2 \quad b4 \quad b5}_{< n5}$ $\underbrace{b3 \quad b6}_{> n5}$

Partition [Heart of Quick sort]

Qu: Given arr[n], rearrange it [s = 0, e = n-1]

arr[0] should go to its sorted position

All el <= arr[0] goto left side of arr[0]

All el > arr[0] " right " " "

arr[] = { 3 1 4 2 5 }

1 2 3 4 5 [valid]

2 1 3 4 5 [valid]

2 1 3 5 4 [valid]

4 1 3 2 5 [invalid]

Brute force:

	0	1	2	3	4	5	6	7	8	9	10
arr[11] = [	10	3	8	15	6	12	2	18	7	15	14]

sort(arr)

$\underbrace{2 \quad 3 \quad 6 \quad 7 \quad 8}_{<=10} \quad \boxed{10} \quad \underbrace{12 \quad 14 \quad 15 \quad 15 \quad 18}_{>10}$

correct position

TC: O(n logn)

SC: O(1)

Approach:

TC: O(n)

SC: O(1)

7

	0	1	2	3	4	5	6	7	8	9	10
arr[11] =	~~10~~	3	8	~~15~~	6	~~12~~	~~2~~	18	~~7~~	15	14
	2	P1	P1	P1	P1	P1	P2	P2	P2	P2	P2
						~~2~~ 10	12		15		
						P2	P1		P2		

left | smaller elements

greater / bigger elements

P1 > P2 [Break]

$\underbrace{2 \quad 3 \quad 8 \quad 7 \quad 4}_{<=10}$ 10 $\underbrace{12 \quad 18 \quad 15 \quad 15 \quad 14}_{>10}$

Happy :- arr[P1] <= arr[0]

arr[P2] > arr[0]

Ex2:

arr[] =	~~14~~	6	8	~~18~~	~~1~~	19	17	~~11~~	20	18	33	29
	1	P1	P1	P1	P1	P1	P2	P2	P2	P2	P2	P2
				11				18				
					P2	P2						
					14							

swap

$\underbrace{1 \quad 6 \quad 8 \quad 11}_{<=14}$ 14 19 $\underbrace{17 \quad 18 \quad 20 \quad 18 \quad 33 \quad 29}_{>14}$

```
void reArrange( int[] arr) {
    int n = arr.length;
    int p1 = 1; p2 = n-1;
    while ( p1 <= p2 ) {
        if ( arr[0] > arr[p1] ) {
            p1 ++;
        } else if ( arr[0] <= arr[p2] ) {
            p2--;
        } else {
            swap( arr, p1, p2);
            p1 ++;
            p2 --;
        }
    }
    swap( arr, 0, p2);
}
```

TC: O(n)

SC: O(1)

* Rearrange subarray [s, e] and return the correct position of first el of subarray.

	0	1	2	3	4	5	6	7	8	9
arr:	10	3	~~8~~	6	~~14~~	~~7~~	~~4~~	12	7	1
			7		4	8	12			

p1 p1 p1 p2 p2

p2

p1

s = 2

e = 7

7	6	4	8	14	12

$\underbrace{7 \quad 6 \quad 4}_{<=8} \quad 8 \quad \underbrace{14 \quad 12}_{>8}$

int reArrange(int[] arr, int s, int e) {

int n = arr.length;

int p1 = ~~1~~ s+1; p2 = ~~n-1~~ e;

while (p1 <= p2) {

if (arr[~~0~~ s] > arr[p1]) {

p1 ++;

} else if (arr[~~0~~ s] <= arr[p2]) {

p2--;

} else {

swap(arr, p1, p2);

p1 ++;

p2 --;

}

}

swap(arr, ~~0~~ s, p2);

return p2;

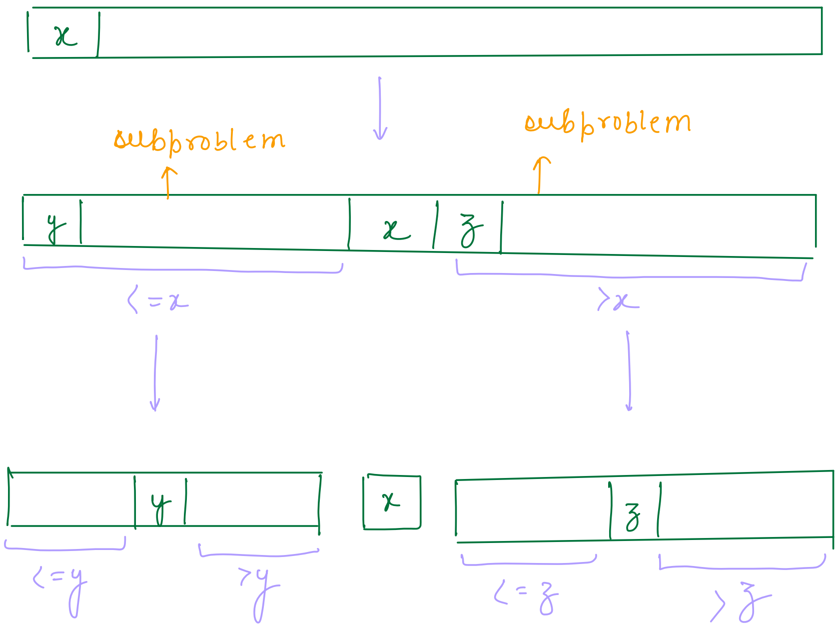
}

TC: O(n)

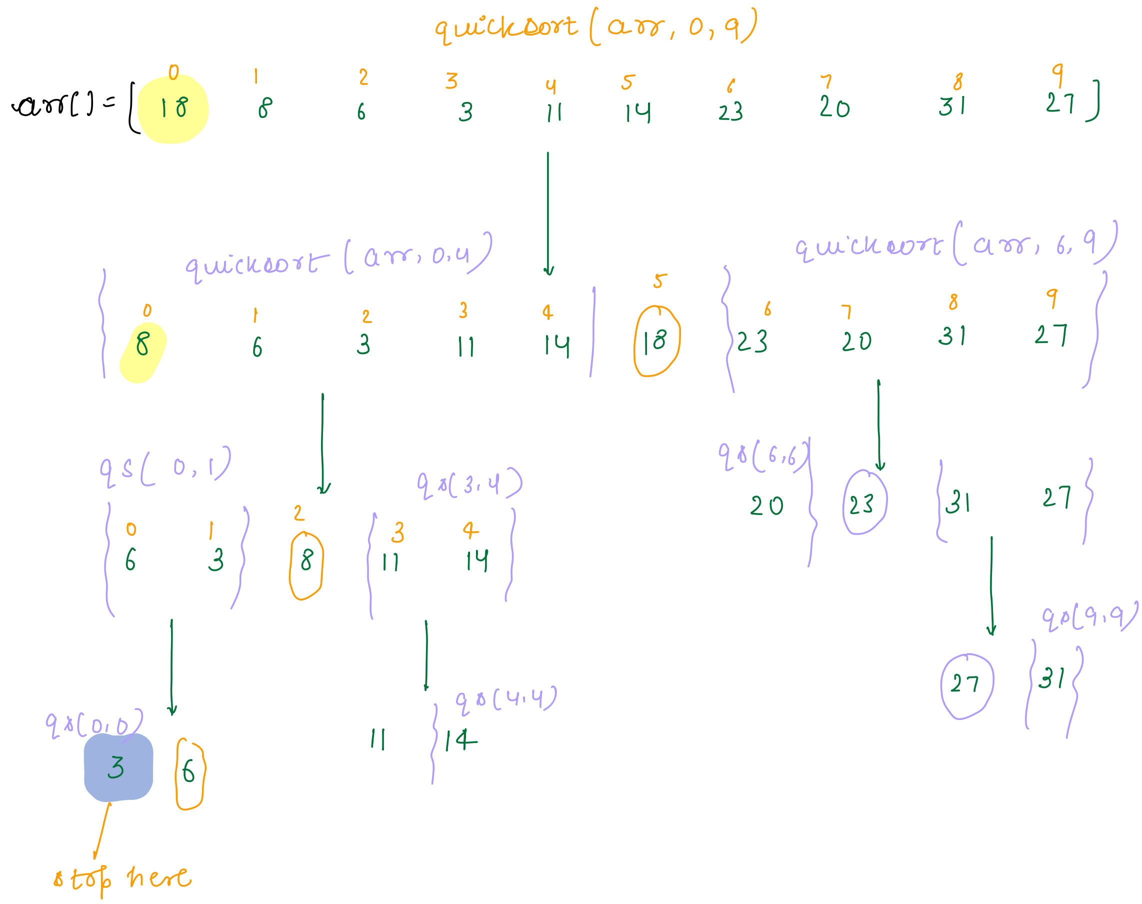
SC: O(1)

Quick sort

arr:



Break: 8:20 AM



⇒ 3 6 8 11 14 18 20 23 27 31.

```
void quicksort(int[] arr, int s, int e) {
    if ( s >= e) {
        return;
    }
    int pIdx = rearrange(arr, s, e);
    quicksort(arr, s, pIdx-1);
    quicksort(arr, pIdx+1, e);
}
```

s == e base case

s > e [cautionary check]

Quicksort [Arrays]

Work : Partition

Recursion

[Preorder]

Mergesort [Linked list]

Recursion

Work : Merge both sorted arrays.

[Postorder]

Time complexities

Case 1: Best time complexity

arr[] = | x | |

<= x | x | > x

middle

Master theorem

$T(n) = 2T\left(\frac{n}{2}\right) + O(n).$

$T(n) = O(\log_2 n)$

n

n/2 n/2

Partition

n

O(n) $\frac{n}{2}$

O(n) $\frac{n}{4}$

O(n) $\frac{n}{8}$

$\frac{n}{16}$

1

$\log_2 n * [n]$

Case 2: Worst case TC.

arr[5] = {3 6 8 9 14}   Sorted array – inc^r manner.

③ 6 8 9 14

[ ]   {6, 8, 9, 14}

{ }   ⑥ {8, 9, 14}

{ }   ⑧ {9, 14}

{ }   ⑨ {14}

Stop

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

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

$T(n) = O(n^2)$

n — $O(n)$

$\downarrow$

$n-1$ — $O(n)$

$\downarrow$

$n-2$ — $O(n)$

$\downarrow$

$n-3$

$\downarrow$

$n-4$

$\vdots$

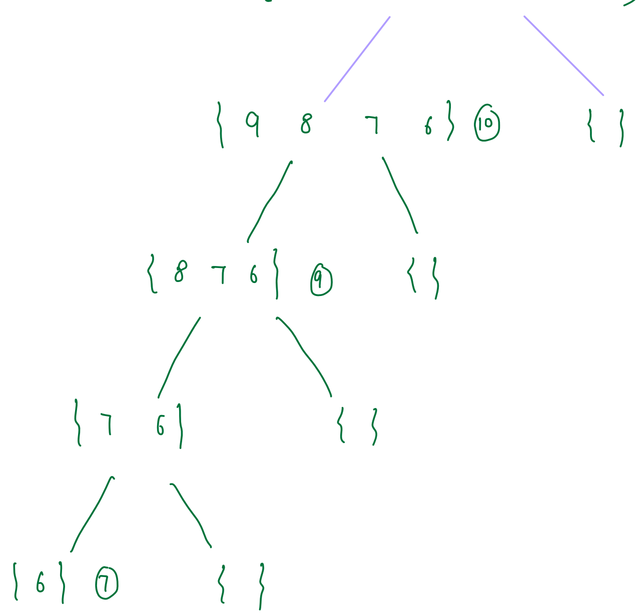
1

$O(n)$

TC: $O(n^2)$

Decreasing array

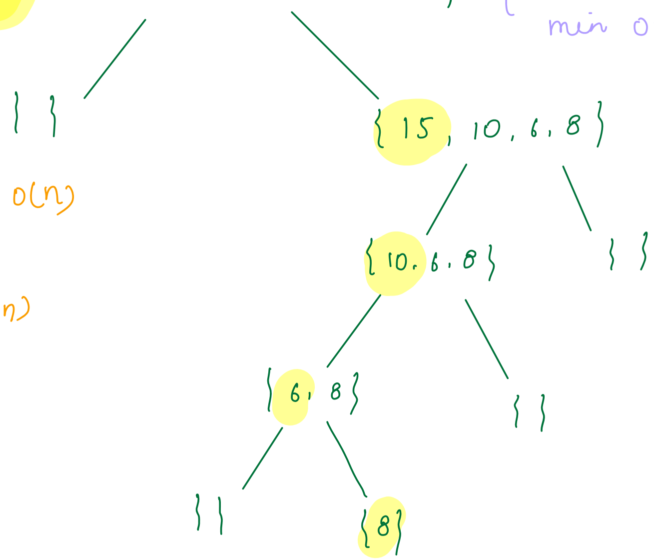
arr[] = [10 9 8 7 6]



TC: $O(n^2)$

Ex3: arr: { 3 15 10 6 8 }

[Reference el is always min or max]



n(5) — $O(n)$

n(4) — $O(n)$

n(3)

n(2)

n(1)

$O(n)$

TC: $O(n^2)$

Problem: If my 0th el is always min | max, TC: $O(n^2)$

↓

Idea: Instead of picking 0th el, I can pick a random element.

	0	1	2	3	4	5	6
arr:	9	6	8	2	10	11	14

2 6 8 9 10 11 14

0th el :- 100% of cases when min | max el as reference

randomel: Decreasing the probability of having min | max el as reference

arr[100] :- $P(\min) = \frac{1}{100}$

$P(\max) = \frac{1}{100}$

$P(\text{not } \min|\max) = \frac{98}{100}$

⇓

Randomised Quick sort ⟶ VVVI ★★★

int reArrange(int[] arr, int s, int e) {

int randomIdx = random(s, e);

swap(arr, 0, randomIdx);

int n = arr.length;

int p1 = ~~1~~ s+1; p2 = ~~n-1~~ e;

while (p1 <= p2) {

if (arr[~~0~~ s] > arr[p1]) {

p1 ++;

} else if (arr[~~0~~ s] <= arr[p2]) {

p2--;

} else {

swap(arr, p1, p2);

p1 ++;

p2 --;

}

}

swap(arr, ~~0~~ s, p2);

return p2;

}

TC: O(n)

SC: O(1)

Quicksort

Avg TC: $O(n \log n)$

SC: $O(\log n)$

Worst TC: $O(n^2)$

SC: $O(n)$

n
|
n-1
|
n-2
|
n-3
⋮
1

$O(n)$

n — $O(1)$ space
|
$\frac{n}{2}$ — $O(1)$ space
|
$\frac{n}{4}$
|
$\frac{n}{8}$
|
$\frac{n}{16}$
⋮
1

$\log n$

Mergesort:

TC: $O(n \log n)$

SC: $O(n)$

Q Why Quick sort preferred?

└ Probability of avg case scenarios will always be much higher than worst case scenarios.

Avg TC / SC of quick sort better Avg TC / SC of merge sort

Mathematical explanation [optional]

arr[] = [1 2 8 7 3 - - - - — . . . . 98 62 100]

array having el 1 to 100 in unsorted manner.

Best reference el :- { 50, 51 } ⇒

Worst reference el: { 1, 100 } ⇒ 2/100 = 0·02

= 0·98 Avg case scenario

n [100] reference is min

{ }   n { 99 }

1 2 3 - - - 100

n [100] reference is smin

{ 1 }   { 3 . . . . 100 } [98]

n = 100

< 90% of el   < 90% of el

1. Reference = 10 ✗

10

{ 9 el }   { 90 el }

< 90% of el   = 90% of el

2. Reference = 91 ✗

91

90 el   9 elements

= 90% of el

Avg reference pt: [11 — 90] = 80 elements

probability of selecting a pivot/reference el that makes sure that there are less than 90% of el on either side is 80%

Thankyou ☺

Doubts

$a = 9 \rightarrow 5+4$ { famous }

$3+2+4$ [α]

$9 = 5+4$

$= 2+3+4$

$2+(2+1)$ $(2+2)$

$4+(4+1)$

$\boxed{num} = [a + b + c + d + e \cdots q]$

$a \quad a+1 \quad a+2 \quad a+3 \; --- \; a+x$

num = form of a & x

$\boxed{a = \text{form of num } \& \ x}$

sqrt

$\boxed{num = (a) + (a+1) + (a+2) --- (a+x)}$

num = $x * a$ + ... { form of a and x }

a = { form of num & x } $\quad 1 - x$

co-efficient of x

a = form of num + $\boxed{\text{coeft of } x}$ * x

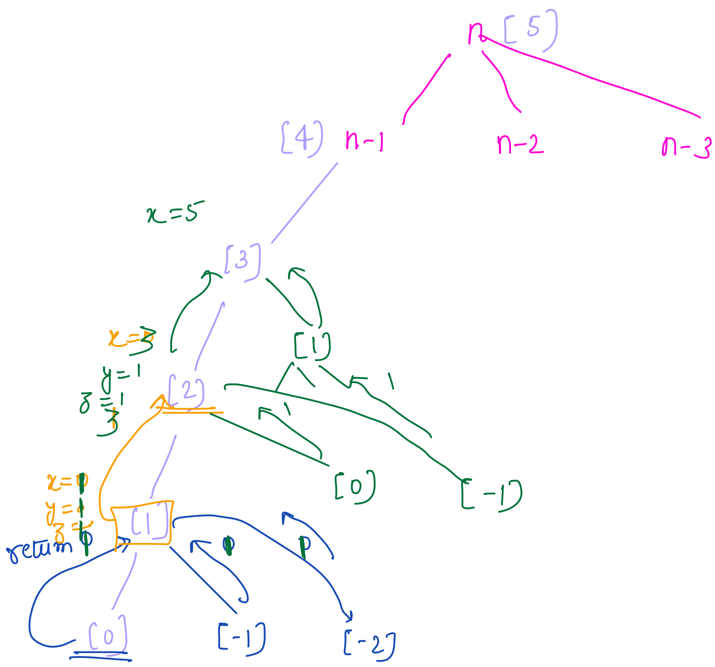
loop

$15 = 1+2+3+4+5$

$= 4+5+6$

$= 7+8$

p & combinatorics




```
fun(n) {
    if(n<=0){ return 1;}
    x=fun(n-1);
    y=fun(n-2);
    z=fun(n-3);
    return x+y+z
}
```