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

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

$\Rightarrow T(n-3) + 2(n-2) + 2(n-1) + 2(n) - 1 - 1 - 1$

$T(n-k) + 2(n-k+1) + \cdots 2(n) - (1)(k)$

At $n-k = 1$ base case arises.

$\Rightarrow T(1) + 2(0+1+2+\cdots n-1) - k + 2n$

$\Rightarrow T(1) + 2\left(\frac{n(n-1)}{2}\right) + 2n - k$

$\Rightarrow T(1) + n^2 - n + 2n - k - 1$

$n = k+1 \Rightarrow k = n-1$

$\Rightarrow T(1) + n^2 + n - n + 1$

$\Rightarrow T(n) = T(1) + n^2 + 1$

If we solve upto $T(n-k)$ where $n-k = 0$,

$\Rightarrow T(0) + 2(0+1+\cdots n) - (k)$

$\Rightarrow 0 + 2\left(\frac{n(n+1)}{2}\right) - k \quad [k = n]$

$\Rightarrow n^2 + n - n = n^2$ $\therefore$ Answer = n^2 option A

2. The correct option is

(c) The asymptotic analysis focuses on the order of growth, focusing on the contribution of leading ~~term~~ in polynomial.

Option(a): There are recurrence relations which cannot be solved by using masters Theorem. Ex: recurrence relation of fibonacci's series (recursive)

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

Option b: For example, in Insertion Sort, the number of computations and swaps differs with the nature of input so it is hard to provide a precise estimate.

optiond: ~~Counter~~ Counter Example $f(n) = n^{1+\sin n}$ and $g(n) = n$ Cannot be compared because $\sin(n)$ is periodic.

3. The correct option is

option (a)

For every iterative algorithm, there is an equivalent recursive algorithm. Converse is false.

Example: There exists a ~~iterative~~ recursive alternative for ~~printing~~ fibonacci series but ~~do~~ if we consider DFS algorithm, only a recursive algorithm exists.

4. ~~Which of~~

The statement of option (b) is false

Option (a) Iterative Fibonacci Sequence : $O(n)$ polynomial

Recursive Fibonacci Sequence : $O(2^n)$ exponential.

Option (b) : $T(n) = T(n-1) + 1$

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

$= T(n-3) + 1 + 1 + 1$

$\vdots$

$T(n-k) + 1 + 1 + \cdots$ k times

At $n-k = 1$ program terminates and $T(1) = 1$

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

$\therefore$ The time complexity for factorial in both iterative and recursive algorithms is $O(n)$.

So option (b) is false as it says the the time complexity for recursive algorithm is exponential.

5. Answer is option (a).

Consider a set A of cardinality 10.

The number of subsets possible for that set is $2^{|A|} = 2^{10}$.

Now to list all subsets, the algorithms has to create and display all 2^{10} subsets (2^n is the size of the input set was 'n'), Therefore, the Algorithm to list all subsets of a set of size 10 has only exponential time complexity and no polynomial time algorithms.

6.

```
main()
{
    printf("enter an integer");
    scanf("%d", &a);
}
```

The time complexity of the following code can be written as:-

option (a)

Both printf and scanf statements are executed in O(1) time complexity.

So combined, we can say that the time complexity can written as $f(n) = \Theta(g(n))$ where $g(n) = 1$ united

9. Contd. $f(n) \leq c(g(n))$ $\quad c > 0$

$f(n) \leq c(1)$ ~~$f(n) = 2$ (two sets~~

printf Statement : $f_1(n) \leq c_1(1)$. — (1)

Scanf statement : $f_2(n) \leq c_2(1)$ — (2)

(1)+(2) $\Rightarrow$ $F(n) \leq (c_1 + c_2)(1)$

$F(n) \leq c(1)$ where $c = c_1 + c_2 \mid c > 0$

We can also say that $f(n) \leq F(n)$

Since $F(n)$ involves two operations with asymptotically equivalent time complexities. This results in

$\Rightarrow$ $c_1(g(n)) \leq F(n) \leq c(g(n))$

where $g(n) = 1$

$\Rightarrow$ $F(n) = \Theta(1)$ (Asymptotically tight bound)

Now consider $h(n) = \log(n)$

If we look at the growth rates, $\log n$ has a higher growth rate asymptotically.

$\Rightarrow$ $F(n) < c \cdot \log n$ $\quad c > 0 \quad n \geq n_0 \quad n_0 > 0$

$\Rightarrow$ $F(n) = o(\log n)$

According to the definition of little oh notation.

7) The recurrence that represents the listing of all permutations of a set of size-'n'.

Ans) (b) ~~$T(n) = T(n-1) + n!$~~ (a) $T(n) = n * T(n-1) + n$

Reason: Let 'S' be the set of size-'n'.

Let $T(n)$ denote the time it takes to list all the permutations of set-size 'n'.

$T(n-1)$ denote for set-size -$(n-1)$.

for each element in 'S'; it takes $T(n-1)$ to generate all the permutations of remaining $(n-1)$ elements. So we have n-choices in selecting 1st element. Combining both of them will give

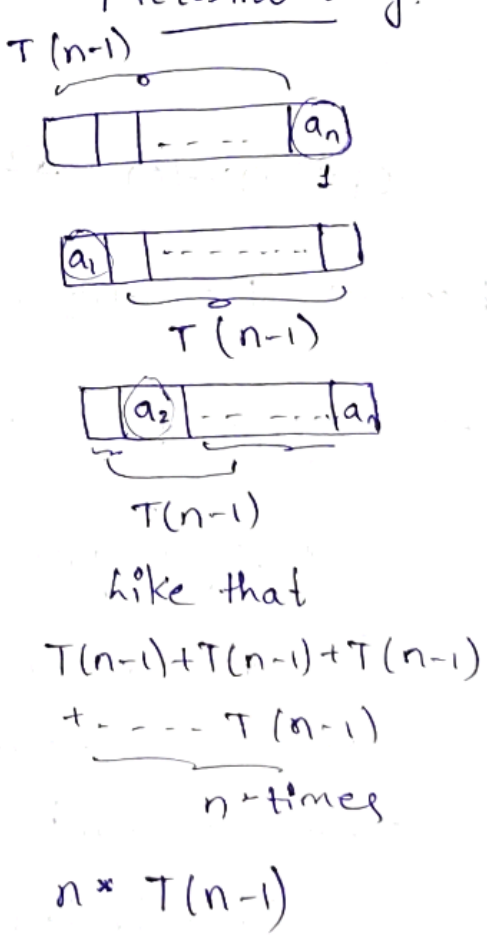
$$T(n) = n * T(n-1)$$

Let printing each list take '1' unit of operation; then,

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

$$\boxed{T(n) = n * T(n-1) + n}$$

Pictorial way:



like that

$T(n-1) + T(n-1) + T(n-1) + \dots + T(n-1)$

(n-times)

$\Rightarrow$ $n * T(n-1)$

8) A problem has six different algorithms $\log\sqrt{n}$, $4^{\log_2 n}$, $n\sqrt{n}$, $2^{n\log n}$, n^n, $\log n$. Arrange them in non-decreasing order of complexity.

Ans) Firstly: $4^{\log_2 n} \longrightarrow n^{\log_2 4} = n^2$,

$2^{n\log n} \longrightarrow 2^{\log n^n} = n^n$,

$\log\sqrt{n} \longrightarrow \frac{1}{2}\log n$,

$\Rightarrow$ $\boxed{\log\sqrt{n}, \log n, n\sqrt{n}, 4^{\log_2 n}, n^n, 2^{n\log_2 n}}$

option-(c)

9) The asymptotic complexity of step count = $n^2\log n + (3.001)^n \log n + n^{2.1} - 17$.

A-) option-(c) $\Omega(n^2\log n)$, $\omega(n^{2.1})$, $o(4^n)$.

Reason: the stepcount function $f(n) = n^2\log n + (3.001)^n \log n + n^{2.1} - 17$.

$\Rightarrow$ $f(n)$ is $\Omega(n^2\log n)$ which means:

$n^2\log n$ grows no-fastes than $f(n)$ as $f(n)$ do have $n^2\log n$ terms but all exponential & polynomial terms present in it So, no matter what constant we take (>0) $\exists\, n_0$ s.t $\forall n \geq n_0$ $f(n) = \Omega(n^2\log n)$. similarly for $n^{2.1}$ also; for all constants 'c' ($c>0$) $f(n) = \omega(n^{2.1})$ as $f(n)$

has exponential term in it which is more dominating than all.

$f(n) = O(4^n)$ as; $4^n > n^2 \log n + (3.001)^n \log n + n^{2.1} - 17$

$\forall n \geq n_0$.

as 4^n is exponential & as n- gets bigger;

$(3.001)^n \log n$ would be the dominating factor;

& which less than 4^n no matter what you add to $f(n)$ & n- changes and keeps on getting bigger. $\Rightarrow$ $\exists n_0 > 0$, $\forall c > 0$, $\forall n \geq n_0$

$\Rightarrow c \cdot 4^n > n^2 \log n + (3.001)^n \log n + n^{2.1} - 17$

10) The time complexity of an algorithm is given by $\{n^2$ if n is even, n if n is odd$\}$. The asymptotic notation is ?

Ans) Best- case: when; n is odd;

the time complexity is $O(n)$

Worst- case: when n is even;

the time complexity is $O(n^2)$

But there isn't common bound; seperate bounds for separate scenario's So, I think theta notation is computable.

So; option- (b) $O(n^2)$, $\Omega(n)$, Theta cannot be computed

11) The sol^n to the recurrence $T(n) = 4T(n/2) + n^2 \log n$ is?

Ans)) $n^{\log_b a} = n^{\log_2 4} = n^2$

$f(n) = n^2 \log n \Rightarrow \frac{n^{\log_b a}}{f(n)} = \frac{n^2}{n^2 \log n} = \frac{1}{\log n}$

$\Rightarrow$

$\frac{f(n)}{n^{\log_b a}} = \frac{n^2 \log n}{n^2} \neq \Omega(n^\varepsilon) \quad \frac{1}{\log(n)} \neq \Omega(n^\varepsilon)$

cannot compare the poly growth of

n^2 & $n^2 \log n$

So, master's thm cannot be applied

Using recurrence we get $T(n) = \Theta(n^2 \log n \log n)$

12) The solution to the recurrence $T(n) = 2^n T(n/6) + 4^n$.

Ans) $n^{\log_b a} = n^{\log_6 2^n} = n^{n \log_6 2} = n^{n.(0.39)}$

$f(n) = 4^n$

$\Rightarrow n^{(0.39)(n)}$

$\Rightarrow \frac{n^{\log_6 a}}{f(n)} = \frac{n^{(0.39)n}}{4^n} = \left(\frac{n^{0.39}}{4}\right)^n \neq \Omega(n^\varepsilon)$

$\frac{f(n)}{n^{\log_6 a}} = \frac{4^n}{n^{(0.39)n}} = \left(\frac{4}{n^{0.39}}\right)^n \neq \Omega(n^\varepsilon)$

cannot compare the poly growth so;

master's thm is not applicable;

(13) $T(n) = T(\sqrt{n}+5) + 2^n$

Let $n = m^2$

$T(m^2) = T(m+5) + 2^{2m}$

$T(m+5) = T(\sqrt{m+5}+5) + 2^{(\sqrt{m+5})^2}$

$T(\sqrt{m+5}+5) = T(\sqrt{\sqrt{m+5}+5}) + 2^{(\sqrt{m+5}+5)^2}$

$T(n) = T(2^k\sqrt{n}+5) + 2^{(2^k\sqrt{n}+5)^2}$

$2^k\sqrt{n} \leq 1$

$\Rightarrow k \geq \log\log n.$

$T(n) = T(2^{\log\log n}\sqrt{n}+5) + 2^{(2^{\log\log n})^2}$

The asymptotic time complexity would be $\theta(\sqrt{n}\log n)$ (d).

14) For first loop:

```
int i=1;n;
while (i<=n) i= i* 3;
```

This loop iterates until i becomes larger than n.
The value of i is multiplied by 3 in each iteration.

$$\Rightarrow 3^k \leq n$$
$$k \leq \log_3 n$$

So, the smallest power of 3 that is grater than or equal to n is $\log_3 n$ which is also the No. of iterations

Similarly for second loop,
Number of iterations required are $\log_k n$

The overall time complexity of code is

$$\theta(\log_3 n) + \theta(\log_k n) \quad (a)$$

15) a) $o(g(n)) \cap \omega(g(n)) = \phi$

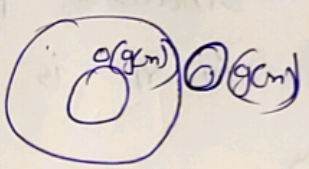
$o(g(n))$ → functions that grow strictly slower than $g(n)$

$\omega(g(n))$ → functions that grew strictly faster than $g(n)$

hence their intersection is empty.

b) $o(g(n)) \subset O(g(n))$ – True.

By defination, o have strict conditions than O.

also, by venn diagram 

c) $o(g(n)) \cap \theta(g(n)) = \phi$

$o(g(n))$ & $\theta(g(n))$ represent different sets of functions.

$o(g(n))$ → functions that grew strictly slower than $g(n)$

$\theta(g(n))$ → functions that grew at same rate of $g(n)$

So, intersection is empty

d) $O(g(n)) \cap \Omega(g(n)) = \theta(g(n))$

$O(g(n))$ → upper bound → functions that grew no faster than $g(n)$

$\Omega(g(n))$ → functions that grew no slower than $g(n)$

then intersection gives $\theta(g(n))$

(a)

16) (b) $T(n) = 2T(n/2) + n \cdot n^{1+\sin n}$

Let, $f(n) = n \cdot n^{1+\sin n}$

check if $f(n)$ satisfies regularity condition of Masters theorm.

$$n \cdot n^{1+\sin n} \leq a \cdot f\left(\frac{n}{b}\right) = a \cdot \left(\frac{n}{2}\right)^{1+\sin\left(\frac{n}{2}\right)}$$ for some $a<1$, $n>>1$

This inequality doesn't hold because the function $n^{1+\sin n}$ is highly oscillatory. So we cannot apply masters theorem directly here.

17) $T(n) = aT(n/b) + f(n)$ $a, b > 1$

$\Omega(f(n)) \rightarrow f(n)$ is lower bound.

$\Omega(n^{\log_b^a}) \rightarrow$ lower bound based on masters theorm

Case 2

$\omega(n^{\log_b^a - \varepsilon}) \rightarrow$ function grows strictly faster than $n^{\log_b^a - \varepsilon}$ for $\varepsilon > 0$.

($f(n)$ may grow much faster than $n^{\log_b^a}$) (a)

18) Consider a problem P and there were two Algorithms to solve P. The complexity of A_1 is $100n^2$ and A_2 is $0.0001\,2^n$.

(a) A_2 is asymptotically faster than A_1, for all inputs whose size greater or equal to 30.

Reason: for $n_0 = 30$

$0.0001\,2^n = 10^{-4} \times 2^{30} = 10^{-4} \times 1073741824$

$100 \times n^2 = \boxed{90000}$ ↪ $\boxed{107374.1824}$

clearly $(0.0001\,2^n) > (100 \times n^2)$

and as LHS is exponential & RHS is polynomial

⇒ exponential grows faster than polynomial.

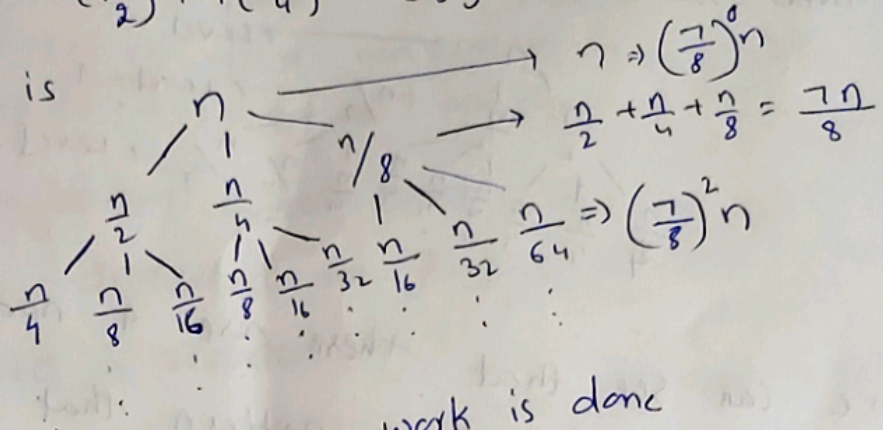
we can say that A_2 is asymptotically faster than A_1. for $\forall n \geq n_0$; $n_0 > 0$

The answer is a) $O(n)$ but not $\Theta(n)$

19) Given reccurrence relation is

$$T(n) = T(\tfrac{n}{2}) + T(\tfrac{n}{4}) + T(\tfrac{n}{8}) + n$$

reccurrence tree is



at level 0 $\Rightarrow \left(\frac{7}{8}\right)^0 n$ units of work is done

at level 1 $\Rightarrow \left(\frac{7}{8}\right)^1 n$ units of work is done

at level k $\Rightarrow \left(\frac{7}{8}\right)^k n$ units of work is done.

Total no. of level is determined by how many times you can divide n by 2.

$$\frac{n}{2^k} = 1 \Rightarrow k = \log n$$

$$\text{Total time} = \left[\left(\frac{7}{8}\right)^0 + \left(\frac{7}{8}\right)^1 + \cdots \left(\frac{7}{8}\right)^{\log n}\right] n \qquad \text{GP sum} = \frac{a(r^n - 1)}{r - 1}$$

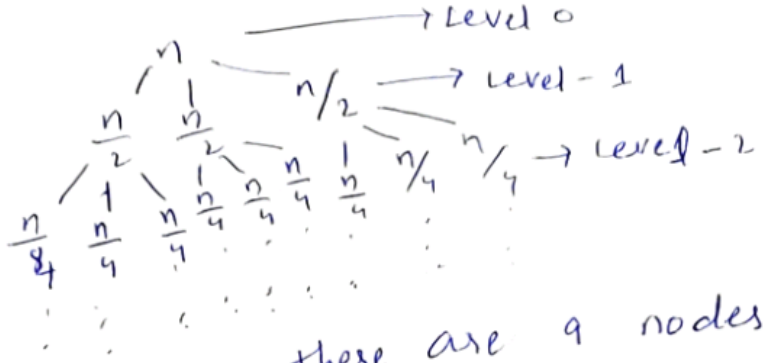
$$= n\left[\frac{1\cdot\left(\left(\frac{7}{8}\right)^{\log n + 1} - 1\right)}{\frac{7}{8} - 1}\right] = 8n\left[1 - \left(\frac{7}{8}\right)^{\log n + 1}\right]$$

then term $\left(\frac{7}{8}\right)^{\log n + 1}$ tends to zero as n becomes larger as $\frac{7}{8} < 1$

So, total time is $8n \Rightarrow O(n)$ but we cannot say $\Theta(n)$ as we don't know the lower bound of the given reccurrence relation.

20) The answer is (b) 9, $\frac{1}{3}$(no. of leaves)

Given recurrence relations is $T(n) = 3T(\frac{n}{2}) + n$



we can see that there are 9 nodes at level 2 also, we can find a pattern that no. of nodes at each level are increased by 3 multiple of Previous level hence, no. of nodes at last but one level = $\frac{\text{no. of node at last level}}{3}$.

21) The answer is b) $n-1$ & $\frac{n}{2}-1$

Given Input is $1, 2, \ldots \frac{n}{2}, n, n-1, \ldots \frac{n}{2}+1$ $(n \in Z)$

In any pass the no. of Comparisons in a bubble sort doesn't change it is $n-1$ only hence $x = n-1$

The Swap occurs only when ~~a[j] and a[j+~~

$a[j] > a[j+1]$ is true

The above Condition will be true $\left(\frac{n}{2}+1\right)^{th}$ position to n^{th} position

The Swap occurs $= n - \left(\frac{n}{2}+1\right) = n - \frac{n}{2} - 1 = \frac{n}{2} - 1$ times

22) The answer is (c) (n-1, 1, 2)

Given x = no. of passes to find minimum.
y = no. of passes to find maximum
z = no. of passes to find 2nd maximum

In bubble sort algorithm after every pass the max element in that pass is deposited at end. & the size of array to be sorted is reduced by 1.

hence to find maximum element only 1 pass is required

to find second maximum element 2 passes are required

on the other hand minimum is found in last pass

i.e, (n-1) passes are required.

23) The answer is (b) Both insertion sort and bubble sort takes $\theta(n^2)$ in worst case.

To find K^{th} maximum (K is a fixed Integer) bubble sort takes $O(nK)$ time

i.e, In worst case if K==n the bubble sort takes $\theta(n^2)$

Insertion sort takes $\theta(n^2)$ to find K^{th} maximum in every case.

Hence, both bubble and Insertion sort take $\theta(n^2)$ time complexity in worst case.

24) (a) ~~The~~ Insertion sort takes atleast $n^2/2$ swaps while sorting

(d) is incorrect as sorted order has no inversion pairs and it would not requiire any swaps and therefore it is not best case input.

(c) is incorrect because as insertion sort is dependent on no. of inversion pairs.

(b) is incorrect no. of comparisons are not dependent on no. of inversion pairs.

a is correct because the condition inside while loop is ~~sta~~ satisfied atleast $\frac{n^2}{2}$ times.

25) Suppose we wish to use binary search module to identify the right position for a[i] as a[1...(i-1)] is already sorted. Then, what is the time complexity of insertion sort.

(a) $T(n) = T(n-1) + \log n$

(b) $T(n) = T(n-1) + \log n + O(1)$

(c) $T(n) = T(n-1) + n \log n$

(d) $T(n) = T(n-1) + \log n + O(n)$

For this question for every element we need to do binary search to get the right position and this takes the time complexity nlogn for the whole elements then for shifting of elements it takes linear time.

So Solving option (b)

$$T(n) = T(n-1) + \log n + O(1)$$

$$T(n) = T(1) + \log n + \log(n-1) + \cdots + \log(1) + n(O(1))$$

$$= \log(n(n-1)\cdots(1))$$

$$= \log(n^n) + n\,O(1)$$

$$T(n) = n \log n + n$$

So Second option is correct.

26) Consider this hybrid sorting; the first $n/2$ elements are sorted using bubble sort and the second half using insertion sort. Suppose the given array is in almost sorted order, then what is the overall time complexity of hybrid sorting

(a) $O(n)$ for this input, for other inputs $O(n^2)$

(b) $O(n^2)$ for this and all inputs

(c) $O(n)$ for all inputs

(d) None of the above

Bubble Sort algorithm compares every element with others so both worst case ~~and best case~~ is $\Theta(n^2)$ in this question we are doing the same but the given array is almost sorted. So with minimal changes it can be sorted so bubble will take $O(n/2)$ and same for insertion sort $O(n/2)$

So the time complexity of this hybrid sorting is $O(n)$

27) the worst case complexity of stable in-place sorting is

(a) $\Theta(n \log n)$ time, $O(n)$ space

(b) $\Theta(n^2)$ time, $O(n)$ space

(c) $\Theta(n \log n)$ time and space

(d) $\Theta(n^2)$ time and space

Stable in-place sorting means for equal elements will be properly arranged according to their input and in-place sorting means it won't use any additional space and all the changes will be done in the original array given.

So examples are bubble sort, insertion sort, of worst time complexity $\Theta(n^2)$ and additional space $O(1)$ and so space is $O(n)$

28) The objective is to find k smallest elements in an array using quick sort partition routine as a black box. then

(a) $O(k)$ calls with overall time complexity $O(n^2)$

(b) $O(n)$ calls with overall time complexity $O(n^2)$

(c) $O(k)$ calls with overall time complexity $O(nk)$

(d) $O(k)$ calls with overall time complexity $O(n\log n)$

we can compare the pivot element with last element of small window and first element of large window then we can get the numbers elements the sort accordingly to find the element this takes $O(k)$ calls with time complexity $O(n\log n)$

a) $T(n) = 2T(n/2) + O(n) + O(n)$

29) ~~d) None of the above~~

The worst case time complexity of quick sort is $T(n) = 2T(n/2) + O(n)$ because at each step we will take $O(n)$ to find median. So at each step two subarrays are created because of the median. So at worst case it takes

$T(n) = 2T(n/2) + O(n) + O(n)$

30) a) All of the above

Option (a) is correct because Quick sort doesn't compare elements but only with pivots. So it is unstable sorting

option (b) If the array is sorted and if we take partition as last element then it would take $O(n^2)$ as every elements before partition is small window and takes $(n-1)$ comp and next take $(n-2)$ comp so it takes $O(n^2)$

option (c) Element in small window will only be compared with partition and same goes for large window so those elements wont be compared.