HOMEWORK 1 SOLUTION

Author- Aarya Kankipati, UIN 01211068

1.

a)

```
NEXT-PERMUTATION(N, A)
for i= N-1 till 0
        if A[i] > A[i-1]:
           target = A[i-1]
           swap = i-1
           while swap<N-1:
               if (A[swap+1]) <= target:
                   break
               swap ++
// i. first non-ascending number from end

// j first number greater than nums[i-1]

// We'll swap these two numbers

           A[swap], A[i-1] = A[i-1], A[swap]
           A[i:] = A[i:][::-1]
           return
// We'll reverse A by traversing in the opposite order
For i = N till 0
    Temp_A.insert(0, A[N-i])

// Setting the value of A to Temp_A
A = Temp_A
return
```

b)

Time Complexity: O(N)

The worst case running time will be depend on input. In the worst case, the first step of NEXT-Permutation() takes O(n) time.

Consider 4 5 3 2 1 as example and need to first locate the first occurrence of an element that is less than the character immediately after it. We may accomplish this by searching backwards through the string. It will require five comparisons. Let us call the left index m, which gives us m=0. Now we must identify the character that is bigger than m after index m in the list. We can do this again by going backwards through the list. That required five comparisons. Let us name that index l, which gives us m=0 and l=1. Swap the two values now. Now, from m+1 to the end of the list, reverse the sequence. If you're working with an array, reversing the items is a linear process. This has no effect on the efficiency of the algorithm.

c)

```
class Solution:
    def nextPermutation(self, list: List[int]):
```

```
for i in range(len(list)-1, 0, -1):
    if list[i] > list[i-1]:
        target = list[i-1]
        swap = i-1
        while swap<len(list)-1:
            if (list[swap+1]) <= target:
                break
            swap += 1
        list[swap], list[i-1] = list[i-1], list[swap]
        list[i:] = list[i:][::-1]
        return

list.reverse()
return
```

2.

In order to prove n^k is $O(2^n)$, for all values of k, Consider functions $f(n)= n^k$,$g(n)= 2^n$

For f(n)=O(g(n)), there will be a real constant c exists, i.e., C>0

F(n)<=c g(n)

As $f(n)= n^k$, $g(n)= 2^n$

$$\text{Limit}_{\text{n-infinity}}\ n^k/2^n = 0$$

$$0< n^k/2^n<=1$$

For large number n, $n>=n_0$

$$0< n^k<=2^n \text{ , for all n}$$

Therefore, $n^k=O(2^n)$ with constant c=1

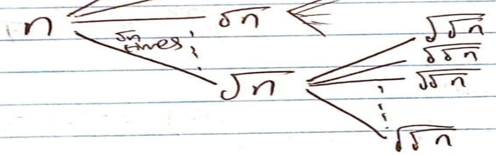
3.

Question-3

Considering the idea for Sorting a list of size n where it needs to be split the list into $\sqrt{n}$ lists. of size $\sqrt{n}$ each of smaller list Individually and then merge the sorted lists to get a sorted list (Single).

Here we are applying divide and Conquer

Dividing n into $\sqrt{n}$ lists of $\sqrt{n}$ size then divide each $\sqrt{n}$ ~~b~~ size from list into $\sqrt{n}$ lists of $\sqrt{\sqrt{n}}$ size



So,, The Running Time will be

$$T(n)=\sqrt{n}\left(T\left(\frac{n}{\sqrt{n}}\right)\right)+n$$

($\because$ as we know, $n=\sqrt{n}\times\sqrt{n}$)

$$T(n)=\sqrt{n}\left(T(\sqrt{n})\right)+n$$

Recurrence relation

$$T(n) = \sqrt{n}\left(T(\sqrt{n})\right) + n$$

Consider $n = p^{2^q}$

So, $T(p^{2^q}) = p^{2^{q-1}} T(p^{2^{q-1}}) + p^{2^q}$ — (1)

By dividing with p^2 on both sides

$$T(p^{2^{q-1}}) = p^{2^{q-2}} T\left(p^{2^{q-2}}\right) + p^{2^{q-1}} \text{ — (2)}$$

(2) in (1) =>

$$T(p^{2^q}) = p^{2^{q-1}}\left(p^{2^{q-2}} T(p^{2^{q-2}}) + p^{2^{q-1}}\right) + p^{2^q}$$

$$= 2p^{2^q} + p^{3 \cdot 2^{q-2}} T \cdot (p^{2^{q-1}})$$

In general,

$$T(2^{p^q}) = 1(p^{2^q}) + p^{(2^t - 1)(2^{q-t})} T(p^{2^{q-t-1}})$$

Consider $q = 1$

$$T(p^{2^q}) = (q+1)(p^{2^q})$$

$$= n(\log(\log n + 1))$$

~~As we know~~

As per our consideration

$$n = p^{2^q}$$

Apply log on both sided

$$\log_p n = 2^q \log_p P$$

$$\log_p n = 2^q (1) \qquad (\because \log_a a = 1)$$

$$\log_p n = (2^q)$$

Apply log on both sides

$$\log_2 (\log_p n) = q \log_2 2$$

$$q = \log (\log n)$$

So, Recurrence relation will be

$$T(p^{2^q}) = \Theta(n(\log(\log n)))$$

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

$$T(n) = \Theta(g(n))$$