1.

Let

$$p(n) = \sum_{i=0}^{d} a_i n^i ,$$

where $a_d > 0$, be a degree-d polynomial in n, and let k be a constant. Use the definitions of the big O, Ω, Θ notations to prove the following properties:

a. If $k \geq d$, then $p(n) = O(n^k)$.

b. If $k \leq d$, then $p(n) = \Omega(n^k)$.

c. If $k = d$, then $p(n) = \Theta(n^k)$.

Solution:-

1. Solution:-

For $p(n) = O(n^d)$. we need to pick $c = a_d + b$. Such that

$$\sum_{i=0}^{d} = a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n + a_0 \leq c n^d$$

dividing it by n^d we get,

$$c = a_d + b \geq a_d + \frac{a_{d-1}}{n} + \frac{a_{d-2}}{n^2} + \dots + \frac{a_0}{n^d}$$

where,

$$b \geq \frac{a_{d-1}}{n} + \frac{a_{d-2}}{n^2} + \dots + \frac{a_0}{n^d}$$

If we choose $b = 1$, then,

$$n_0 = \max(d a_{d-1}, d\sqrt{a_{d-2}}, \dots, d\sqrt[d]{a_0})$$

Now, we have n_0 and c such that,

$p(n) \leq c n^d$ for $n \geq n_0$ is definition of $O(n^d)$.

$\therefore p(n) = O(n^k)$. (for $k \geq d$.)

For $p(n) = \Omega(n^d)$, we need to pick $c = a_d + b$ such that,

$$\sum_{i=0}^{d} = a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n + a_0 \geq c n^d$$

dividing it by n^d we get,

$$c = a_d + b \le a_d + \frac{a_{d-1}}{n} + \frac{a_{d-2}}{n^2} + \cdots + \frac{a_0}{n^d}$$

where,

$$b \le \frac{a_{d-1}}{n} + \frac{a_{d-2}}{n^2} + \cdots + \frac{a_0}{n^d}$$

If we choose $b = -1$, then,

$$n_0 = \max(d\,a_{d-1}, d\sqrt{a_{d-2}}, \ldots, d\sqrt[d]{a_0})$$

Now we have n_0 and c_2 such that,

$p(n) \ge c_2 n^d$ for $n \ge n_0$ is defination of $\Omega(n^d)$

$\therefore p(n) = \Omega(n^k)$ (for $k \le d$)

As per above. Now we have n_0, c_1 and c_2 such that,

$$c_1 n^d \le p(n) \le c_2 n^d$$

for $n \ge n_0$ is defination of $\theta(n^d)$.

$\therefore p(n) = \theta(n^k)$ (for $k = d$)

2.

Show that for any real constants a and b, where $b > 0$,

$$(n + a)^b = \Theta(n^b) .$$

Solution:-

Solution-2:-

Let constant a, b, where $b > 0$.

$$(n+a)^b \le (2n)^b \quad , \text{ for } n \ge |a|$$
$$= 2^b n^b$$
$$= c_1 n^b \quad , \text{ where } c_1 = 2^b$$

So, $(n+a)^b$ is $O(n^b)$ ...... (1).

$$(n+a)^b \ge (n/2)^b \quad , \text{ for } n \ge 2|a|$$
$$= 2^{-b} n^b$$
$$= c_2 n^b \quad , \text{ where } c_2 = 2^{-b}$$

So, $(n+a)^b$ is $\Omega(n^b)$ ...... (2)

The result follows from (1) & (2) with,

$c_1 = 2^b$, $c_2 = 2^{-b}$, $n_0 \ge 2|a|$

Therefore, $(n+a)^b = \Theta(n^b)$.

3. Implement the brute force algorithm of the max subarray problem. You algorithm should have running time of O(n_2).

Solution:-

```
max_profit=0
n=A.length
for i=0 to n-1
        for j=i+1 to n
                diff=A[j]-A[i]
                if diff>max_profit
                        max_profit=diff
                        buy=i
                        sell=j
```

Here, we have 2 for loop which have running time as below other than that all other statement has constant running time,

$(n-1)+(n-2)+\ldots+(1) * (n-1)+(n-2)+\ldots+(1) = O(n^2)$

4. Implement the linear algorithm for max subarray problem explained in lecture.

Solution:-

```
class Max
{
    public static int maxeg(int[] A)
    {
        int maxSoFar = A[0];
        int maxEndingHere = A[0];

        for (int i = 1; i < A.length; i++)
        {
            maxEndingHere = maxEndingHere + A[i];
            maxEndingHere = Integer.max(maxEndingHere, A[i]);
            maxSoFar = Integer.max(maxSoFar, maxEndingHere);
        }
        return maxSoFar;
    }

    public static void main(String[] args)
    {
        int[] A = { -8, -3, 6, 2, -5, 4 };
        System.out.println("The sum of contiguous subarray with the " +
                                        "largest sum is " + maxeg(A));
    }
}
```

5. Use divide-and-conquer technique to find the max value of an input integer array. Assuming we divide at the middle of the array to create two subproblems each time.

a. Write the pseudocode of the algorithm.
b. Write the recursive running time equation.
c. Find out the running time from this recursive equation.
d. What is a better way to divide so that the running time is lower?

Solution:-

```
Max(I,j,max)
if(i=j)
        max=a[i]
elseif(i=j-1)
        if(a[i]<a[j])
                max=a[j]
        else
                max=a[i]
else
        mid=((i+j)/2)
        max(i, mid, maxl)
        max(mid+1, j, maxr)
        if(maxl<maxr)
                maxl=maxr
```

Recursive running time equation for above pseudocode is,

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

$T(1) = 0$

$T(2) = 1$

To find the running time from above recursive equation we need to take n=k,

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

Let n=k,

$$\begin{aligned}
T(k) &= 2^k T(n/2^k)+2+2^2+\ldots+2^k \\
&= 2^k T(n/2^k)+2(2^2-1) && \text{(According to } S_n=[a(r^n-1)]/[n-1]) \\
&= (2^{k+1}/2)T(n/2^k)+2((2^{k+1}/2)-1) \\
&= (2^{\log_2 n}/2)T(2^{k+1}/2^k)+2((2^{\log_2 n}/2)-1) && (n=2^{k+1} \Rightarrow \log_2 n=\log_2 2^{k+1}) \\
&= (n/2)T(2)+2((n/2)-1) \\
&= (n/2)T(2)+n-2 \\
&= (n/2)(1)+n-2 \\
&= 3(n/2)-2
\end{aligned}$$

There is no other better way to lower running time.