Q1 $\begin{cases} T(n) = aT(\frac{n}{b}) + f(n) \\ f(n) = n^d \log^k n \end{cases}$

(a) $T(n) = 2T(n/3) + 1$

1. $a=2 \quad b=3 \quad d=0 \quad k=0$

2. $\log_b a = \log_3 2 \quad > d=0$

3. $T(n) = \Theta(n^{\log_3 2})$

(b) $T(n) = 5T(n/4) + n$

1. $a=5 \quad b=4 \quad d=1 \quad k=0$

2. $\log_b a = \log_4 5 \quad > d=1$

3. $T(n) = \Theta(n^{\log_4 5})$

(c) $T(n) = 9T(n/3) + n^2$

1. $a=9, \quad b=3, \quad d=2, \quad k=0$

2. $\log_b a = \log_3 3^2 = 2 = d=2$

3. ~~matt~~ $T(n) = \Theta(n^2 \log n)$

(d) $T(n) = 8T(n/2) + n^3$

1. $a=8, b=2, d=3, k=0$
2. $\log_2 8 = d$
3. ~~null~~ $T(n) = \Theta(n^3 \log n)$

(e) $T(n) = 49T(n/25) + n^{3/2}\log n$

1. $a=49, b=25, d=3/2, k=1$
2. $\log_{25} 49 \leq d$
3. $\Theta(n^{3/2}\log n)$

Q2

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

depth

cn^2

$c(\frac{n}{2})^2$ $\quad$ $c(\frac{n}{2})^2$ $\quad \rightarrow (\frac{1}{4})cn^2$

$c(\frac{n}{4})^2$ $c(\frac{n}{4})^2$ $c(\frac{n}{4})^2$ $c(\frac{n}{4})^2$ $c(\frac{n}{4})^2$ $c(\frac{n}{4})^2$ $\rightarrow (\frac{1}{4})^2 cn^2$

$\vdots$

$\Theta(1)$ ~~~~~~~~~~ $\Theta(1)$. $\rightarrow \Theta(n)$

$2^{\log_2 n} = n^{\log_2 2} = n$

1. Depth = $\log_2 n$

2. The total cost at any depth i

$\sum_{i=0}^{\log_2 n} (\frac{1}{4})^i cn^2$

3. The number of leaves:

$2^{\log_2 n} = n^{\log_2 2} = n$

The total cost:

$\Theta(n)$

4. $T(n) = \sum_{i=0}^{\log_2 n} (\frac{1}{4})^i cn^2 + \Theta(n)$

$< \sum_{i=0}^{\infty} (\frac{1}{2})^i cn^2 + \Theta(n)$

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

$= 2(n^2 + \Theta(n))$

$= O(n^2)$

Q3

```
selection sort(int[] array) {                                   times
    int index = 1 ;                                              1
    int len = n ;                                                1
    while(index < n-1) {                                         n-1
        int min = array[index];                                  n-2
        int minindex = index ;                                   n-2
        for(int j = index+1; j < n; j++) {                       Σ_{i=2}^{n} ti
            if(array[j] < min) {                                 Σ_{i=2}^{n} (ti-1)
                min = array[j] ;                                 Σ_{i=2}^{n} (ti-1)
                minindex = j ;
            }
        }
        int temp = array[minindex]                               n-2
        array[minindex] = array[i]                               n-2
        array[i] = temp;                                         n-2
    }
}
```

Worst Case:

The outer loop runs n-1 times. So the each iteration of the outer loop, the inner loop runs:

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

Thus, $\Theta(n^2)$ is the worst time complexity.

Statement:

1. **Define a Function:**

Create a function findRotationIndex(arr, left, right) to recursively search for the rotation index.

Base Case:

- If the subarray has only one element, return its index since it is the minimum.

Check if Subarray is Already Sorted:

- If arr[left] <= arr[right]
- it means the array segment is already sorted. In this case, arr[left] is the smallest element, and its index is the rotation index.

Calculate the Middle Index:

- Compute mid = left + (right - left) / 2.

Compare Middle with its Neighbors:

- If arr[mid] > arr[mid + 1], then mid + 1 is the rotation index (smallest element).
- If arr[mid] < arr[mid - 1], then mid is the rotation index (smallest element).

Decide Which Side to Search:

- If arr[mid] >= arr[left], the left part is sorted, so search the right half.
- Otherwise, search the left half.