

I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of 0 for the involved assignment for my first offense and that I will receive a grade of F for the course for any additional offense.

1. Use the Master theorem to solve the following recurrences.

- (a) $T(n) = 3T(n/4) + n$
 $a = 3, b = 4, f(n) = n$
Case 3: $f(n) = \Theta(n^c)$ if $c = 1$.
 $\log_4 3 = 0.79248 < c$
 $T(n) = \Theta(f(n)) = \Theta(n)$
- (b) $T(n) = 2T(n/4) + \sqrt{n} \log(n)$
 $a = 2, b = 4, f(n) = \sqrt{n} \log(n)$
Case 2: $f(n) = \Theta(n^c \log^k n)$ if $c = \frac{1}{2}$ and $k = 0$
 $\log_4 2 = 0.5$ so $c = \log_b a$
 $T(n) = \Theta(n^{0.5} \log^1 n) = \Theta(\sqrt{n} \log(n))$
- (c) $T(n) = 5T(n/2) + n^2$
 $a = 5, b = 2, f(n) = n^2$
Case 1: $f(n) = \Theta(n^c)$ if $c = 2$
 $\log_2 5 = 2.3219... > c$
 $T(n) = \Theta(n^{\log_2 5}) = \Theta(n^{2.3218...})$

2. Solve the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{for } n \leq 1 \\ T(n/4) + T(3n/4) + n & \text{otherwise} \end{cases}$$

using the recursion tree method. Draw the recursion tree and show the aggregate instruction counts for the following levels (0th, 1st, and last levels), and derive the Θ growth class for $T(n)$ with justifications.

3. Use the substitution method to prove that $T(n) = T(n-1) + n \in O(n^2)$

Proof. Assume that $T(n) = O(n^2)$. So then $T(n) \leq c \cdot n^2$ for some constant c . Assume $T(k) \leq ck^2$ for $k < n$. Prove $T(n) \leq cn^2$ by induction.

$$\begin{aligned} T(n) &= T(n-1) + n \leq c \cdot (n-1)^2 + n \\ &\leq c \cdot (n-1)(n-1) + n \\ &\leq c \cdot (n^2 - 2n + 1) + n \\ &\leq cn^2 - cn + c \leq cn^2 \end{aligned}$$

Which holds provided $cn + c \geq 0$. Which is $cn \geq -c$. So $T(n)$ is in $O(n^2)$ as long as $c \geq 0$ and $n \geq 0$. \square

4. Assume that you are given an array of n ($n \geq 1$) elements sorted in non-descending order. Design a *ternary* search function that searches the array for a given element x by applying the divide and conquer strategy.

- **Divide:** Grab an array index at $1/3$ of the array length (a_1) and at $2/3$ of the array length (a_2). That way the indexes split the array into thirds.
- **Conquer:** If the element x is less than $A[a_1]$ then it must be in the subarray $A[0 \text{ to } a_1]$. Otherwise if x is greater than $A[a_1]$ and less than $A[a_2]$ then it must be in the subarray $A[a_1 \text{ to } a_2]$ Lastly if x is greater than $A[a_2]$ then it must be in the subarray $A[a_2 \text{ to } n]$. Then recursively search the subarray until x is the value of $A[a_1]$ or $A[a_2]$.
- **Combine:** The final answer is the index found when the recursive function returns.

```
function ternarySearch(x, A, left, right)
    a_1 = 1/3 * (right-left) // first index
    a_2 = 2/3 * (right-left) // second index
    if A[a_1] == x return a_1 // found x
    if A[a_2] == x return a_2

    // check left subarray
    if A[a_1] > x return ternarySearch(x, A, left, a_1-1)

    // check right subarray
    else if A[a_2] < x return ternarySearch(x, A, a_2+1, right)

    // check middle subarray
    else return ternarySearch(x, A, a_1+1, a_2-1)
```

The recursive time complexity of ternarySearch would be $T(n) = T(n/3) + \Theta(1)$. $n/3$ because the size of the array that needs to be searched is divided by three. Other functions of ternarySearch is trivial so happens over $\Theta(1)$

5. Develop a divide-and-conquer approach to selection (and hence a solution for the finding median problem). Hint: for any number v , imagine splitting list S into three categories: elements smaller than v , those equal to v (there might be duplicates), and those greater than v .