CS 331: Algorithms and Complexity (Fall 2023)
Unique Number: 52765, 52770

Assignment 4 - Solution

Due on Tuesday, 27 February, by 11.59pm

Problem 1

(10 points)

(a) (1pt each)
$T_2(n)$ has $\frac{4}{3}$ inside the recurrence, while results in subsequent calls growing the value of n.
$T_3(n)$ has $-5n^3$ as the cost, which results in negative time complexity.

(b) (2pt each)

- $T_1(n) = 2T_1(\frac{n}{4}) + n^2$, $T_1(1) = 1$
  Using Master's Theorem, a = 2, b = 4, f(n) = n^2
  $n^{\log_b a} = n^{\log_4 2} = n^{\frac{1}{2}}$
  $f(n) = \Omega(n^{\log_b a})$, so $T_1(n) = \Theta(f(n)) = \Theta(n^2)$

- $T_4(n) = 2T_4(\frac{n}{2}) + n \log n$, $T_4(2) = 0$
  Using Master's Theorem, a = 2, b = 2, f(n) = $n \log n$
  $n^{\log_b a} = n^{\log_2 2} = n$
  However, f(n) = $n \log n$ is not polynomially larger than n, so we cannot use Master's Theorem.
  I will use the recurrence relation

Level	Problem size	Total time
0	n	$n \log n$
1	$\frac{n}{2}$	$2 \cdot \frac{n}{2} \log \frac{n}{2}$
2	$\frac{n}{4}$	$2^2 \cdot \frac{n}{2^2} \log \frac{n}{2^2}$
$\vdots$	$\vdots$	$\vdots$
k	$\frac{n}{2^k} = 1$	$2^k \cdot \frac{n}{2^k} \log \frac{n}{2^k}$

$\rightarrow (\sum_{i=0}^{k-1} 2^i \cdot \frac{n}{2^i} \log \frac{n}{2^i}) + 1 \cdot 2^k$

$\rightarrow (\sum_{i=0}^{k-1} n \log \frac{n}{2^i}) + 2^k$

$\rightarrow (\sum_{i=0}^{log_2(n)-1} n \log \frac{n}{2^i}) + 2^{log_2 n}$

$\rightarrow (\sum_{i=0}^{log_2(n)-1} n \log n - n \log 2^i) + n$

$\rightarrow n + \sum_{i=0}^{log_2(n)-1} n \log n - \sum_{i=0}^{log_2(n)-1} i$

$\rightarrow n + n \log_2 n \log_2 n - \frac{log_2(n)(log_2(n)-1)}{2}$

- $T_5(n) = \sqrt{n}T_5(\sqrt{n}) + n$, $n \geq 2$
  We cannot use Master's Theorem here, as the form of the recurrence is not in the form of $T(n) = aT(\frac{n}{b}) + f(n)$.

Level	Problem size	Total time
0	n	n
1	$n^{1/2}$	$n^{1/2}n^{1/2}$
2	$n^{1/2^2}$	$n^{1/2^2}n^{1/2^2}$
$\vdots$	$\vdots$	$\vdots$
k	$n^{1/2^k} = 2$	$n^{1/2^k}n^{1/2^k}$

(c) (2 pts)

Problem 2

(10 points)

(a) (4 points) Split the array into 5 equal parts, until the size of the array is 1. Then, merge the arrays back together.
To merge the arrays, we will use 5 pointers to each of the 5 arrays, let them be size n/5. We then compare each of them to find the smallest element. Then, we will add the smallest element to the new array of size n, repeat until all pointers are at the end of the array.
This yields a new sorted array of size n.

(b) (2 points) Let each array be size n/5. We assume each of the arrays are sorted.
Each iteration has at most 4 comparisons, so the total number of comparisons is at most 4n.

(c) (2 points) T(n) = 5T(n/5) + O(n) with T(1) = 1

(d) (2 points) Using Master's theorem, a = 5, b = 5, f(n) = n
$n^{log_5 5} = n$.
$n^{log_b a} = f(n)$, so the recurrence is $\Theta(n \log n)$.

Problem 3

(10 pts)

(a) Recursive Solution with Memoization

```
getMaxRecur(i):
    if i > n:
        return 0
    if memo[i] != -1:
        return memo[i]
    MAX = max(p_i + getMaxRecur(i + c_i), getMaxRecur(i + 1))
    memo[i] = MAX
    return MAX
```

(b) Iterative Solution

```
getMaxTabular():
    memo[n] = 0
    for i in range(n, 0):
        memo[i] = max(p_i, memo[i + c_i])
    return memo[1]
```

(c) Proof of Correctness
Since the iterative solution iterates through the array computing each element once, it's $O(n)$.