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	$\frac{n}{2^2}$	$2^2 \cdot \frac{n}{2^2} \log \frac{n}{2^2}$
$\vdots$	$\vdots$	$\vdots$
k	$\frac{n}{2^k} = 1$	$2^k \cdot \frac{n}{2^k} \log \frac{n}{2^k}$

$\rightarrow \sum_{i=0}^{k} 2^i \cdot \frac{n}{2^i} \log \frac{n}{2^i}$

$\rightarrow \sum_{i=0}^{\log_2 n} n \log \frac{n}{2^i}$

$\rightarrow \sum_{i=0}^{\log_2 n} n \log n - n \log 2^i$
$\rightarrow n \log n \log n - \sum_{i=0}^{\log_2 n} ni$
$\rightarrow n \log n \log n - n\frac{\log_2 n(\log_2 n+1)}{2}$
$\rightarrow n \log n \log n - \frac{n \log_2 n \log_2 n}{2} - \frac{n \log_2 n}{2}$
$\rightarrow \frac{n \log_2 n \log_2 n}{2} - \frac{n \log_2 n}{2}$
$\therefore$, $T_4(n) = \Theta(n \log n \log n)$

- $T_5(n) = \sqrt{n}T_5(\sqrt{n}) + n$, $n \geq 2$
  I'll convert this recurrence into a different form to allow for using Master's Theorem
  Let $m = \log_2 n$; $n = 2^m$
  Then, $T_5(n) = \sqrt{n}T_5(\sqrt{n}) + n$ becomes $T_5(2^m) = 2^{m/2}T_5(2^{m/2}) + 2^m$
  Divide both sides by 2^m
  This yields $\frac{T_5(2^m)}{2^m} = \frac{T_5(2^{m/2})}{2^{m/2}} + 1$
  We define $S(m) = \frac{T_5(2^m)}{2^m}$
  Now, the equation is equivalent to $S(m) = S(m/2) + 1$
  Using Master's Theorem, a = 1, b = 2, f(m) = 1
  $f(m) = \Theta(1)$
  $m^{\log_b a} = m^{\log_2 1} = m^0 = 1$
  $f(m) = \Theta(m^{\log_b a})$, so the recurrence is $\Theta(\log m)$
  Then, $S(m) = \Theta(\log m)$
  Since $S(m) = \frac{T_5(2^m)}{2^m}$, then $S(m)2^m = T_5(2^m)$
  Then, $T_5(2^m) = \Theta(2^m \log_2 m)$
  Then, $T_5(n) = \Theta(n \log n \log n)$

(c) (2 pts)
Both T_4 and T_5 are $\Theta(n \log n \log n)$
They are smaller than T_1, which is $\Theta(n^2)$

Problem 2

(10 points)

(a) (4 points)
Divide: Split array into 5 equal parts of size n/5.
Combine: Merge the size n/5 sorted arrays back together into a size n sorted array.
Split the array recursively 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 their respective array.
This yields a new sorted array of size n.
We combine the sorted arrays to get the final sorted array.

(b) (2 points)
Let each array be size n/5. We assume each of the arrays are sorted.
Each element has at most 4 comparisons until some subarrays have no elements, so the total number of comparisons is at most 4n - $\sum_{i=1}^{5} i$ = 4n - 10.
Example: [1, 6], [2, 7], [3, 8], [4, 9], [5, 10] — n = 10
We compare 1, 2, 3, 4, 5 $\rightarrow$ 4 comparisons
We compare 6, 2, 3, 4, 5 $\rightarrow$ 4 comparisons
We compare 6, 7, 3, 4, 5 $\rightarrow$ 4 comparisons
We compare 6, 7, 8, 4, 5 $\rightarrow$ 4 comparisons
We compare 6, 7, 8, 9, 5 $\rightarrow$ 4 comparisons
We compare 6, 7, 8, 9, 10 $\rightarrow$ 4 comparisons
We compare 7, 8, 9, 10 $\rightarrow$ 3 comparisons
We compare 8, 9, 10 $\rightarrow$ 2 comparisons
We compare 9, 10 $\rightarrow$ 1 comparison
We compare 10 $\rightarrow$ 0 comparisons
Total comparisons = 30 = 4(10) - 10.

(c) (2 points)
We split the array into 5 equal subarrays recursively so $T(n)$ becomes $5T(n/5)$.
The combining step takes $\Theta(n)$ time.
T(n) = 5T(n/5) + 4n - 10 with T(1) = 1

(d) (2 points)
Using Master's theorem, a = 5, b = 5, f(n) = 4n - 10
$f(n) = \Theta(n)$
$n^{log_5 5} = n.$
$f(n) = \Theta(n^{log_b a})$, so the recurrence is $\Theta(n \log n)$.

Problem 3

(10 pts)

(a) Recursive Solution with Memoization

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

(b) Iterative Solution

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

(c) Proof of Correctness
Claim: MT(low) = OPT(low), in other words, MT returns the maximum points possible with tasks starting at time low
Base Case: low = n
MT(n) = p_n if it's greater than 0, which is the maximum points possible.
Inductive Hypothesis: Assume MT(i) = OPT(i) for all i $\geq$ k.
Inductive Step: Prove MT(k - 1) = OPT(k - 1)
By the inductive hypothesis, we know that MT(k) = OPT(k) and MT(i + c_i) = OPT(i + c_i)
Case 1: It's better to not do the task k - 1
In other words, OPT(k - 1) = OPT(k)
If that's the case, our algorithm picks the maximum of the two options p_{k-1} + MT(k + c_{k-1}) and MT(k), which would yield MT(k)
Then, MT(k - 1) = MT(k)
This yields MT(k - 1) = MT(k)
By the inductive hypothesis, MT(k) = OPT(k), which is equivalent to OPT(k - 1) since the task is not worth taking
Then, MT(k - 1) = OPT(k - 1)
Then, case 1 of MT is optimal
Case 2: It's better to do the task k - 1
In other words, OPT(k - 1) = p_{k-1} + OPT(k + c_{k-1})
If that's the case, our algorithm picks the maximum of the two options p_{k-1} + MT(k

Problem 3 continued on next page...

+ c_{k-1}) and MT(k), which would yield p_{k-1} + MT(k + c_{k-1})
Then, MT(k - 1) = p_{k-1} + MT(k + c_{k-1})
By the inductive hypothesis, MT(k + c_{k-1}) = OPT(k + c_{k-1})
Add p_{k-1} to both sides, we get p_{k-1} + MT(k + c_{k-1}) = p_{k-1} + OPT(k + c_{k-1}) = OPT(k - 1)
Then, MT(k - 1) = OPT(k - 1)
Then, case 2 of MT is optimal
Therefore, MT(k - 1) = OPT(k - 1)
Therefore, MT(low) = OPT(low)
Therefore, the iterative solution is correct and optimal.
Since the iterative solution iterates through the array indexing each element once, it's $O(n)$.