CSCI 2300: Introduction to Algorithms

Homework 4

Lucien Brule
Prof. Bulent Yener
April 27, 2023

1 Problem 1

Problem: Compute tight big-Oh bounds for the following recurrences:
a. $T(n) = 8T\left(\frac{n}{4}\right) + O(n)$
b. $T(n) = 2T\left(\frac{n}{4}\right) + O(\sqrt{n})$
c. $T(n) = T(n-4) + O(n^2)$
d. $T(n) = T(\sqrt{n}) + O(n)$

Part A:

Using the Master Theorem, we have $a = 8$, $b = 4$, and $f(n) = O(n)$. Thus, we get $\log_b a = \log_4 8 = 3/2$. Since $n^{\log_b a} = n^{3/2}$ and $f(n) = O(n)$, we are in case 1 of the Master Theorem. Therefore, the tight big-Oh bound is:

$$T(n) = O(n^{\log_b a}) = O(n^{3/2})$$

Part B:

Using the Master Theorem, we have $a = 2$, $b = 4$, and $f(n) = O(\sqrt{n})$. Thus, we get $\log_b a = \log_4 2 = 1/2$. Since $n^{\log_b a} = \sqrt{n}$ and $f(n) = O(\sqrt{n})$, we are in case 2 of the Master Theorem. Therefore, the tight big-Oh bound is:

$$T(n) = O(n^{\log_b a} \log n) = O(\sqrt{n} \log n)$$

Part C:

For this recurrence, we can use the iterative method:

$$T(n) = O(n^2) + T(n-4) = O(n^2) + O((n-4)^2) + T(n-8) = \cdots$$

After k iterations, we have:

$$T(n) = O(n^2) + O((n-4)^2) + \cdots + O((n-4k)^2) + T(n-4k)$$

Since we can have at most $\frac{n}{4}$ iterations, the tight big-Oh bound is:

$$T(n) = O\left(\sum_{k=0}^{\frac{n}{4}} (n-4k)^2\right) = O(n^3)$$

Part D:

For this recurrence, we can use the substitution method. Let $m = \log_2 n$, so $n = 2^m$. Then, the recurrence becomes:

$$S(m) = S(m-1) + O(2^m)$$

Solving this recurrence, we get:

$$S(m) = O(2^m) + O(2^{m-1}) + \cdots + O(2^1) = O(2^m)$$

Substituting back $n = 2^m$, we have:

$$T(n) = O(n)$$

2 Problem 2

Problem: Let A be an array of n integers, and let R be the range of values in A, i.e., $R = \max(A) - \min(A)$. Give an $O(n + R)$ time algorithm to sort all the values in A.

Solution:

We can use the Counting Sort algorithm to sort the values in A in $O(n + R)$ time. The algorithm is as follows:

1. Find the minimum and maximum values in A, i.e., $\min(A)$ and $\max(A)$.
2. Calculate the range R: $R = \max(A) - \min(A)$.
3. Initialize an array C of size $R + 1$ with all elements set to 0.
4. Count the occurrences of each element in A and increment the corresponding entry in C.
5. Calculate the cumulative sum of C.
6. Create a new array B of size n for the sorted output.
7. Iterate through A in reverse order, and for each element, place it in the correct position in B using the values from C, and decrement the corresponding entry in C.

The following is the pseudo code for the Counting Sort algorithm:

```
function counting_sort(A):
    min_A = min(A)
    max_A = max(A)
    R = max_A - min_A
```

```
C = [0] * (R + 1)

for a in A:
    C[a - min_A] += 1

for i in range(1, len(C)):
    C[i] += C[i - 1]

B = [0] * len(A)

for a in reversed(A):
    B[C[a - min_A] - 1] = a
    C[a - min_A] -= 1

return B
```

The time complexity of this algorithm is $O(n+R)$, as each step takes at most $O(n)$ or $O(R)$ time.

3 Problem 3

Problem: Let A be an array of n distinct integers. Consider an algorithm to find the minimum value, where we pair up the elements, and retain the smaller of the values from each pair. This will result in an array of half the size (actually the resulting size will be $\lfloor \frac{n}{2} \rfloor$). We can then recursively apply the same approach, until we get a final array with just two elements. We compare these two values and return the minimum of those values as the answer. Answer the following questions:
a. How many comparisons are done in the above algorithm in the worst case.
b. Show how to modify/extend this method to find the second smallest element.
c. Prove that we can find the second smallest element in $n + \lceil \log(n) \rceil - 2$ comparisons in the worst case.

Part A:

In the first round, we perform $\lfloor \frac{n}{2} \rfloor$ comparisons. In the second round, we perform $\lfloor \frac{n}{4} \rfloor$ comparisons, and so on. The total number of comparisons can be calculated as follows:

$$\sum_{i=1}^{\lceil \log_2(n) \rceil} \left\lfloor \frac{n}{2^i} \right\rfloor$$

As we sum to $\lceil \log_2(n) \rceil$, the worst-case scenario is when n is a power of 2. In that case, the expression becomes:

$$\sum_{i=1}^{\log_2(n)} \frac{n}{2^i} = n \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \right)$$

Since this is a geometric series with a sum of 1, the total number of comparisons in the worst case is:

$$n - 1$$

Part B:

To find the second smallest element, we need to remember the "losers" of each comparison with the smallest element during the tournament. We can store these elements in a separate array or list. After finding the smallest element, we need to find the minimum value among these "losers", which will be the second smallest element.

Part C:

Let's analyze the number of comparisons made to find the second smallest element. In the initial tournament, we made $n - 1$ comparisons to find the minimum value. After that, we need to compare the "losers" with the smallest element. Since there were $\lceil \log(n) \rceil - 1$ rounds of comparisons before reaching the final two elements, we have $\lceil \log(n) \rceil - 1$ "losers" to compare. R Thus, the total number of comparisons needed to find the second smallest element is:

$$(n - 1) + (\lceil \log(n) \rceil - 1) = n + \lceil \log(n) \rceil - 2$$