CSPB 3104 - Park - Algorithms

Started on	Tuesday, 6 February 2024, 11:47 AM
State	Finished
Completed on	Tuesday, 6 February 2024, 12:19 PM
Time taken	31 mins 57 secs
Marks	20.00/23.00
Grade	**8.70** out of 10.00 (**87**%)

Question **1**

Complete

Mark 12.00 out of 12.00

We compare two different divide and conquer algorithms for the same problem.

Algorithm A: On an array of size n, the algorithm produces FOUR sub arrays, each of size $\frac{n}{2}$ performing $\Theta(n^2)$ work in the process. The time taken to combine the output of the sub problems is $\Theta(n)$.

Algorithm B: On an array of size n, the algorithm produces THREE sub arrays, each of size $\frac{n}{2}$ performing $\Theta(n^2)$ work in the process. The time taken to combine the output of the sub problems is $\Theta(n)$.

1) Write down the recurrences for the running time A(n) for algorithm A and B(n) for algorithm B on inputs of size n. You do not need to write out the base case.

2) Solve for A(n) using the master method to derive a Theta bound on worst case running time of algorithm A.

3) Solve for B(n) using the master method to derive a Theta bound on worst case running time of algorithm B.

Master Method

The recurrence $T(n) = aT(\frac{n}{b}) + \Theta(n^c)$ with base case

$T(1) = \Theta(1)$ has the solution

Case 1 $\log_b a > c$ then $T(n) = \Theta(n^{\log_b(a)})$

Case 2 $\log_b a = c$ then $T(n) = \Theta(n^{\log_b(a)} \log(n))$

Case 3 $\log_b a < c$ then $T(n) = \Theta(n^c)$

Please note that the quantity $\epsilon = \log_b(a)$ for the recurrences in this problem.

Algorithm A

For this algorithm, we are told that this algorithm produces four sub arrays (in this case a = 4) of size n / 2 (in this case b = 2). The dominant term in the the work step is on the order O(n^2). So this means our recurrence relation is going to be:

$A(n) = 4T\left(\frac{n}{2}\right) + \Theta(n^2)$.

In this case, $\epsilon = \log_2(4) = 2$ and $c = 2$, so in this case, Case 2 of the Master Method applies and the solution will then be:

$\Theta(n^c \log(n)) = \Theta(n^2 \log(n))$

Algorithm B

For this algorithm, we are told that this algorithm produces three sub arrays (in this case a = 3) of size n / 2 (in this case b = 2). The dominant term in the work step is on the order O(n^2). So this means our recurrence relation is going to be:

$B(n) = 3T\left(\frac{n}{2}\right) + \Theta(n^2)$.

In this case, $\epsilon = \log_2(3) \approx 1.5849$ and $c = 2$, so in this case, Case 3 of the Master Method applies and the solution will then be:

$\Theta(n^c) = \Theta(n^2)$.

Comment:

Algorithm A
A(n) = $4A(n/2) + \theta(n^2) + \theta(n)$
Algorithm B
b(n) = $3B(n/2) + \theta(n^2) + \theta(n)$

\theta(n^2 log(n)) --> case 2
\theta(n^2) --> case 3

Question **2**

Incorrect

Mark 0.00 out of 1.00

Algorithm A runs in time $f(n) = 7 \times 2^n + n^2 + n\log(n)$

Algorithm B runs in time $g(n) = 3 \times 2^{1.2n} + n^4 + n^2$

Which algorithm is asymptotically faster?

Select one:

- a. Algorithm A
- b. Algorithm B ✖
- c. They are "equally fast" in an asymptotic sense.

Your answer is incorrect.

The correct answer is: Algorithm A

Question **3**

Correct

Mark 1.00 out of 1.00

Algorithm A runs in time $100000n + 4\sqrt{n} + 1.5\log(n)$

Algorithm B runs in time $200n + 4n^{0.999} + 0.1093\log(5n)$.

Which algorithm is **asymptotically** faster?

Select one:

- a. Both algorithms are asymptotically equal $\Theta(n)$ ✔
- b. Algorithm A
- c. Algorithm B

Your answer is correct.

The correct answer is: Both algorithms are asymptotically equal $\Theta(n)$

Question **4**

Correct

Mark 2.00 out of 2.00

Consider the function below:

```
def dodo(a) :
      n = length(a)
      sum .= 0
      for i = 1 to n :
            if  a[i] == 0:
                        return 0
             for j = i+1 to n
                        sum = sum + a[i] - a[j]
      return sum
```

Select all true statements from the list below. n refers to the size of the input array a.

Select one or more:

- [x] a. The best case running time is $\Theta(1)$ ✔
- [ ] b. The best case running time is $\Theta(n)$
- [ ] c. The worst case running time is $\Theta(1)$
- [ ] d. The worst case is realized on an array whose elements are all 0s.
- [x] e. The worst case running time is $\Theta(n^2)$ ✔

Your answer is correct.

The correct answers are: The worst case running time is $\Theta(n^2)$
, The best case running time is $\Theta(1)$

Question **5**

Correct

Mark 3.00 out of 3.00

A divide and conquer algorithm takes an array of n elements and divides into three sub arrays of size $\frac{n}{4}$ each taking $\Theta(n)$ time to do the subdivision. The time taken to combine outputs of each subproblem is $\Theta(1)$.

Which of the recurrences below will describe the running time $T(n)$ on an input of size n? We ignore the base case for this discussion.

Select one:

a. $T(n) = 4T(\frac{n}{3}) + \Theta(1)$

b. $T(n) = 3T(\frac{n}{4}) + \Theta(n)$ ✔

c. $T(n) = 3T(\frac{n}{4}) + \Theta(1)$

d. $T(n) = 4T(\frac{n}{3}) + \Theta(n)$

e. $T(n) = 4T(n) + \Theta(n^{\frac{3}{4}})$

Your answer is correct.

The correct answer is: $T(n) = 3T(\frac{n}{4}) + \Theta(n)$

Question **6**

Partially correct

Mark 2.00 out of 4.00

Given an array a of size n and value v, we wish to find the element $a[j]$ that is "closest" to v. Mathematically, the closest element is one that minimizes $|a[j] - v|$ the absolute value of the difference between the element and v.

If there are multiple elements in the array that are equally close then any of the elements can be returned.

What is the complexity of the best algorithm to solve this problem? Choose all valid options from those given below.

Select one or more:

- [x] a. If the array a is sorted, then the worst case complexity is $\Theta(\log(n))$ obtained by modifying binary search. ✔
- [ ] b. The best approach is to use insertion sort to sort the array in ascending order and then perform a scan of the array a from left to right.
- [ ] c. We can solve the problem in $\Theta(1)$ time by simply checking if the middle element of the array equals v or not.
- [x] d. If the array a is unsorted, then the worst case complexity is $\Theta(n)$ obtained by scanning the elements of the array one by one. ✔
- [x] e. The best approach is to use merge sort to sort the array in ascending order and then perform a modified binary search algorithm. ✖

Your answer is partially correct.

You have selected too many options.
The correct answers are: If the array a is unsorted, then the worst case complexity is $\Theta(n)$ obtained by scanning the elements of the array one by one.
, If the array a is sorted, then the worst case complexity is $\Theta(\log(n))$ obtained by modifying binary search.