1879B735-45B4-4419-B14F-1DC43BBAA548

winter-2018-cs-341-midterm-41bbd

#337 2 of 34

(Extra space.)

Instructions

- NO CALCULATORS OR OTHER AIDS ARE ALLOWED.
- You should have 34 pages (including the header and extra pages).
- Make sure you fill the information on the header page.
- Solutions will be marked for clarity, conciseness and correctness.
- If you need more space to complete an answer, you may continue on the two blank extra pages at the end.

Useful Facts and Formulas

1. Master Theorem
   Suppose that $a \geq 1$ and $b > 1$, $d \geq 0$. Consider the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta(n^d)$$

   Then:

$$T(n) \in \begin{cases} \Theta(n^{\log_b(a)}) & \text{if } a > b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^d) & \text{if } a < b^d. \end{cases}$$

2. $a^{\log_b c} = c^{\log_b a}$

(Extra space.)

BA43EC06-92DB-48BA-8D07-759C78C01E41

winter-2018-cs-341-midterm-41bbd

#337 5 of 34

Q1ab 8

1. **(17 marks)** *Short Questions 1* For each question below, give your answer together with a brief explanation. Show computations if it is appropriate to do so.

a) **(5 marks)** Suppose $f(n) = O(n)$, $g(n) = O(n)$, then is $f(n) + g(n) = O(n)$? If so, please provide a proof. Otherwise provide a counter example (i.e., an example f(n) and g(n) that are $O(n)$ but $f(n) + g(n)$ is not $O(n)$).

Yes, $f(n)+g(n) \in O(n)$, because of the max rule, when adding functions we only care about the highest growth rate as it will represent the upperbound. Since f(n) and g(n) have the same running time, the highest/max grow rate is $O(n)$.

$\therefore$ $f(n) + g(n) \in O(n)$.

b) **(3 marks)** Order these three functions in increasing order of asymptotic growth rate. You do not need to justify your answer.

$f(n) = \log(n)^{50} + 1.01^n$

$g(n) = \log(n)^{175} + n^{0.550}$

$h(n) = \log(n)^{200} + n^{0.450}$

h(n), g(n), f(n).

(Extra space.)

Q1cd 9

c) **(3 marks)** Give three examples of divide-and-conquer algorithms. Just list the names of the algorithms or the computational problems they solved.

— Merge-Sort → use divide and Conquer to Sort algorithm.

— Saddle-back Search → use divide-and Conquer to Search a 2D matrix.

— Max-Subarray → we use this to find the maxium

d) **(6 marks)** Consider running the Gale-Shapley algorithm on an input with the following intern and hospital preferences:

Interns	Hospital Ranks	Hospitals	Intern Ranks
1	A B C D	A	4 1 2 3
2	B C D A	B	1 4 3 2
3	C D A B	C	2 3 4 1
4	D A B C	D	1 2 3 4

- **(3 marks)** When interns propose, what is the matching given by the GS algorithm?

  (1, A)
  (2, B)
  (3, C)
  (4, D).

- **(3 marks)** When hospitals propose, what is the matching given by the GS algorithm?

(A, 4).
(B, 1).
(C, 2).
(D, 3).

(Extra space.)

FA8EB104-9FC5-4B2F-A962-0678BEEABE99
winter-2018-cs-341-midterm-41bbd
#337 9 of 34

Q2ab 5

2. (13 marks) *Short Questions 2* For each question below, give your answer together with a brief explanation. Show computations if it is appropriate to do so.

a) **(3 marks)** In the job scheduling problem we had in lectures, where each job had both a weight and a length, and the goal is to minimize the weighted total completion time, what was the optimal greedy strategy? (If your answer says "sort by parameter x", please indicate if you're sorting in increasing or decreasing value.)

Sort by. weight in decreasing order.

0+5

b) **(5 marks)** Consider a sequence $F(n)$ defined by the following recursive formula: $F(n) = F(n-1) + F(n-2)$, with $F(0) = 0$ and $F(1) = 1$. Given n, we want to design an algorithm to compute $F(n)$. Explain why an approach based on blind recursion is a terrible idea, and why dynamic programming yields a faster algorithm. We are asking just for a conceptual answer, not an explicit dynamic programming algorithm or its pseudocode, correctness, and runtime analysis.

With blind recursion there will be a lot of repeated calculations of the same value, this will lead to unnecessary computations, thus resulting in a large polynomial running time. Dynamic programming for this problem would yield a faster running time as partial results would be stored in memory so they do not need to be re calculated when needed.

E5D383CA-2503-4F98-B69A-EC7D18BC864B

winter-2018-cs-341-midterm-41bbd

#337 10 of 34

(Extra space.)

1EFD989F-2E2F-4571-8F25-94AF77970BD2
winter-2018-cs-341-midterm-41bbd
#337 11 of 34

Q2c 0

c) **(5 marks)** Recall the matrix multiplication oder (also referred to as paranthesization) problem. Given matrices $M_1, \ldots, M_n$, where M_i has dimension $d_{i-1} \times d_i$, we need to compute the optimal order of computing $M_1 \times \ldots \times M_n$ that minimizes the total number of numeric multiplications needed. Consider the following pseudocode for the dynamic programming algorithm that computes the minimum number of numeric multiplications needed,

procedure DP-Matrix-Ordering($d_0, d_1, \ldots, d_n$):
Base Cases: $S[i][i] = 0$; $S[i][i+1] = d_{i-1}d_i d_{i+1}$
for $i = 1, \ldots, n$
    for $j = i+2, \ldots, n$
        $x = +\infty$
        for $k = i, \ldots, j-1$
            $x = \min(x, S[i][k] + S[k+1][j] + d_{i-1}d_k d_j)$
        $S[i][j] = x$
return $S[1][n]$

What is wrong with this algorithm?

(Extra space.)

Q3 0

3. (10 marks) *Recurrences.* Consider the recurrence:

$$T(n) = 4T(n/2) + 2n$$

$$T(1) = 1$$

Prove $T(n) = O(n^2)$ by induction.

Hint: Guess $T(n) = an^2 - bn$ for some a, $b > 0$.

Base Case. : $n=1$, The running time is finally n^2.

I.H. : Assume, the running time is. n^2 for all $n' < n$,

I.S. Now we must prove $O(n^2)$ for $n' = n$.

we guess $T(n) = an^2 - bn$.

$T(n) = 4T(n/2) + 2n$.

$=$.

$\therefore O(n^2)$.

(Extra space.)

4. (20 marks) *Divide and Conquer.* CHAOS is an international spy organization, which has n agents in separate locations. Each agent has a computer that has an identical copy of an encryption key. An enemy organization attacked CHAOS' system lately, which may have changed the stored keys in some of these computers. Luckily, strictly more than half of the agents' keys are unaffected. CHAOS hired you to find out which agents' keys are unaffected.

a) **(5 marks)** You have access to a system called TEST-SAME that can, through a safe channel, communicate with two agents i and j and tell you if agents i and j have the same key. Given an agent i, design a procedure to determine whether the agent is unaffected with $O(n)$ TEST-SAME operations.

Q4a 5

```
Dertmine. (i, agents).   //assuming agents does not include i
   int count =0;
   for (int j=0; j< agents.length; j++) {
      if ( TESTSame (i, agents (j))) {
         count ++
      }
   }.
   if (count. > (agents.length /2))) return true.
   else return false.
```

(Extra space.)

$2(\frac{n}{2})+n$, $b=2$. $c=1$.

$a=2$ $2=2^1$

Q4b 1

b) **(10 marks)** Design a divide and conquer algorithm, that performs $O(n \log(n))$ TEST-SAME operations to find all of the unaffected agents. Give the pseudocode of your algorithm and give brief explanation of its correctness (you'll do runtime analysis in part (c)).

//Assume determine can be used. from part a.

```
Get unaffected. (int begin, int end, agents ) {
  IF (begin == end) determine (agent(begin), agents) (4)).   IF true output.
  else. {
    mid = (begin + end) / 2.
    Get unaffected (begin, mid)!
    Get unaffected (mid+1, end);
```

Incorrect algorithm. 1

3.

- Case. 1 : Only two agent exist. Then. the solution trivial - just check
- Case 2 ,

(Extra space.)

9D9480F9-130E-49E8-9870-F4BFEA0C381E

Q4c 2

c) **(5 marks)** Write down the recurrence for the runtime of your algorithm and analyze its runtime (i.e, # TEST-SAME operations). You may use the Master Theorem.

running time of part A

$T(n) = 2\left(\frac{n}{2}\right) + n$

$a = 2, \; b = 2, \; d = 1.$

This is not the analysis for the algorithm you gave. And it is T(n/2). 2

$a = 2^1,$ ... of the.

master theorem, So our running time is

$\Theta(n \log n)$ which is in $O(n \log n)$.

F2F3F1B3-49C2-4F23-890B-B9674DA4FC50

winter-2018-cs-341-midterm-41bbd

#337 20 of 34

(Extra space.)

{5,3} {1,-1}

-2, -3, -4, -5

{5, 4, 3} {1, 2, -1}

{20,

(-3, -5) = 8

(-2, -4) = 6

{1, 3} (-2, -4) = 6 (-3, -5) = 8 {10, 20}

{5, 7, 15} {4, 9, 11}

5 - 4 = 1
5 - 9 = 4
5 - 11 = 6

7 - 4 = 3
7 - 9 = 2
7 - 11 = 4

15 - 4 = 11
15 - 9 = 6
15 - 11 = 4

34 - 10 = 24
34 - 20 = 14
34 - 30 = 4

50 - 10 = 40
50 - 20 = 30
~~50 - 30 = 20~~

55 - 10 = 45
55 - 20 = 35
~~55 - 30 = 25~~

5 - 1 = 4
5 - 1 = 5

3 - 1 = 3
3 - (-1) = 4

1 - 10 = 9
1 - 20 = 19

3 - 10 = 7
3 - 20 = 17

(3, -1) = 4
(5, 1) = 4

1, 2 3, 4

1-3=2
1-4=3

2-3=1
2-4=-2

16, 4

10, 2

115 = 4
7, 117 = 10
14

2, 3 = 1
1, 4 = 3

1, 3 = 2
2, 4 = 2

(7, 5) = 2 = 80
(1, 17) = 16
20

5. (20 marks) *Greedy Algorithms.* Consider the following matching problem: Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, ..., b_n\}$ be $2n$ numbers (you can assume for simplicity, they are distinct). Your goal is to match each a_i to one b_j (and vice versa), i.e., form pairs $(a_{i_1}, b_{j_1}), (a_{i_2}, b_{j_2}), \ldots, (a_{i_n}, b_{j_n})$, where $i_1, \ldots, i_n$ and $j_1, \ldots, j_n$ are permutations of $\{1, \ldots, n\}$, such that the following function is *minimized*:

$$\Sigma_{k=1}^{n} |a_{i_k} - b_{j_k}|$$

That is, your goal is to minimize the sum of the absolute values of the differences of pairs. Here are two different greedy strategies to solve this problem:

Strategy 1: Pick the pair (a, b) with the smallest $|a - b|$ value, where $a \in A$ and $b \in B$. Then remove a and b, and repeat.

Strategy 2: Pick the pair (a, b), where a is the smallest number in A and b is the smallest number in B. Then remove a and b, and repeat. Note that we can implement this method in $O(n \log(n))$ time by sorting.

a) **(5 marks)** Give a counterexample showing that one of these strategies is incorrect (i.e., it can sometimes give a non-optimal solution). Consider $A = \{1, 7\}$ $B = \{17, 5\}$.

Q5a 5

$\boxed{S1}$
(7, 5) = 2
(1, 17) = 16
Total = = 18
18 > 14

$\boxed{S2}$
(1, 5) = 4
(7, 17) = 10
Total = 14.

As can be seen, Strategy 1, is not the optimal strategy

(Extra space.)

Q5b 0

b) **(3 marks)** A mathematical fact is if $a < a'$ and $b < b'$, then $|a - b| + |a' - b'| \leq |a - b'| + |a' - b|$. There are 6 possible cases corresponding to the 6 different possible orderings of a, a', b, b'. For example, one ordering is $a < b < b' < a'$. One can prove this fact by considering all 6 cases and showing that the inequality holds in each case. Instead, show only that the inequality holds for the $a < b < b' < a'$ ordering. (You might want to skip this part initially and come back to it after proving part (c)).

F946D53C-EC58-4098-8A87-61114150F846

winter-2018-cs-341-midterm-41bbd

(Extra space.)

c) **(12 marks)** Give a proof that the other strategy is correct (i.e., it always gives an optimal solution).

Hint: Consider an exchange argument that uses the mathematical fact from part (b).

Q5c 2

Assume $G_1, \ldots G_n$ is the greedy solution and $O_1 \ldots O_n$ is the optimal solution. Since we can freely sort the values. also note that greedy algorithm differ by order not in length Assume first $i-1$ pairs are identical. Then we have [illegible] B. If the pairs are not sorted by increasing order. we can modify O by swapping a and b. This will only affect the sum after. A. So we are left with A + sum(a) and A + sum([illegible] [illegible] fact that [illegible] mathematical by part b. [illegible] given [illegible] actually have a lower sum, This is a contradiction since, O was assumed to be otimal.

∴ G is optimal as it selects sums in such a way that individual sums are minimized.

Swap operation not clear

Not strictly worse (only <=), could still be opt. Need a finiteness argument to get to greedy order

BF4F2077-D26F-4FBC-A248-4D74C76BA07A

winter-2018-cs-341-midterm-41bbd

(Extra space.)

B84D22F8-DD46-4B1B-9F90-4ECC89E37247
winter-2018-cs-341-midterm-41bbd
#337 27 of 34

6. **(20 marks)** *Dynamic Programming.* You are driving an electric car to go from Waterloo to Vancouver along a fix route. There are n stations at fixed locations $0 = d_0 < d_1 < \ldots < d_n$ in the middle that can provide *battery changes*. For simplicity $d_0 = 0$ is Waterloo, and d_n is Vancouver.

At station i (at distance d_i) there is a battery B_i that lasts for some L_i distance and costs C_i dollars. At each station i, you can replace your battery with the battery B_i and pay C_i dollars or continue driving with the remaining of the current battery you have. Using a dynamic programming algorithm, compute the minimum cost for completing this trip. At the end of the trip, any extra battery power that is left in the battery is worth nothing. You can also assume that there is a solution to the problem, i.e., after picking the first battery at d_0, there is a sequence of battery swaps that can make the car reach Vancouver).

a) **(10 marks)** Let $D[i]$ be the minimum cost to reach the station at d_i. Give a recurrence relation to compute $D[i]$. Briefly justify the correctness.

Q6a 0

$$D[i] = \begin{cases} D[i-1] & \text{if } d_i - d_{i-1} \\ D[i-1] + B_{i-1} & \text{if } d_{i+1} - d_i > L_{i-1}. \end{cases}$$

The cost of reaching a station is either the previous cost if the user chose not to refill or previous cost + C_i if the user chose to refill, due to not having enough charge. Since extra power is worth nothing, car should only be recharged when empty.

(Extra space.)

7B33CBF1-15D4-41EE-841A-E3891285E1CF

winter-2018-cs-341-midterm-41bbd

#337 29 of 34

b) **(10 marks)** Based on (a), provide the pseudocode for the dynamic programming algorithm for this problem. Analyze the time complexity.

Q6b 0

$D(0) \leftarrow 0$

int length = d_0

For int i = 1 to n.

if (D_{i+1} > Len) D[i] = D[i-1] + B_i

else

Output D(n).

The loop is a basic for loop, therefore its running time is limited to number of times it recurses, therefore it is O(n).

(Extra space.)

(Extra space.)

(Extra space.)

(Extra space.)

347272A4-3E20-4B7C-B2DE-B090D008924A

winter-2018-cs-341-midterm-41bbd

#337 34 of 34

(Extra space.)