German University in Cairo
Faculty of Media Engineering and Technology

Bar code

Midterm Solution

CSEN703: Analysis and Design of Algorithms
Winter 2020 Semester

Dr. Wael Abouelsaadat

Duration: 2 hours

Do **not** turn this page until you have received the signal to start.
In the meantime, read the instructions below carefully.

This exam consists of 4 questions (numbered 1 to 4) on 10 pages *(including this one)*, printed on one side of the paper. When you receive the signal to start, please make sure that your copy of the examination is complete.

Answer each question directly on the examination paper, in the space provided, and **use the reverse side of the page for rough work**. If you need more space for one of your solutions, use the reverse side of the page and indicate **clearly** the part of your work that should be marked.

1. _____ / 20 (Divide and Conquer Algorithm Design)
2. _____ / 20 (Greedy Algorithm Design)
3. _____ / 20 (Dynamic Programming Algorithm Design)
4. _____ / 10 (Asymptotic Analysis)

_____ **/ 70** **TOTAL**

Question 1. Divide and Conquer Design [*20 marks total*]

a) [*6 marks*] Device a divide and conquer algorithm that takes as input two n-element arrays *A* and *B* of numbers, and a value *val*. The algorithm returns true if there are indices *i* and *j* such that $A[i] + B[j] = val$ and false otherwise. You can use pseudo code or code.

Solution

- *Algorithm*
  - sort the array *a* in ascending order and the second array *b* in descending order
  - start from first element in *a* and first element in *b* compare their sum with val
  - if the sum is greater than the value *val*, then move to the next element in *b*, and if the sum is lesser than the *val*, the move to the next element in *a*
  - Keep doing this till the given sum is obtained, else return false

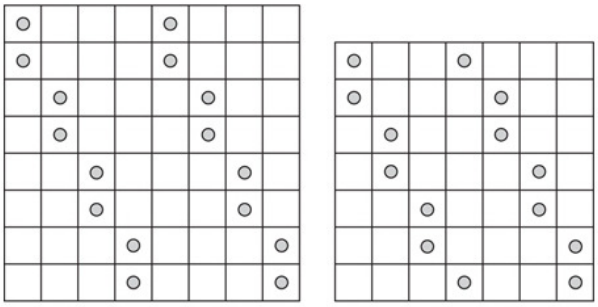
Notes for Marking

- The above is a variation of divide and conquer called reduce and conquer. To correctly reduce the input to reach faster to the solution, you have to do some kind of re-organization to the data (sorting and sorting in reverse as mentioned above).
- We will accept other variations involving modifications to binary search to keep going left and right to find the pair.

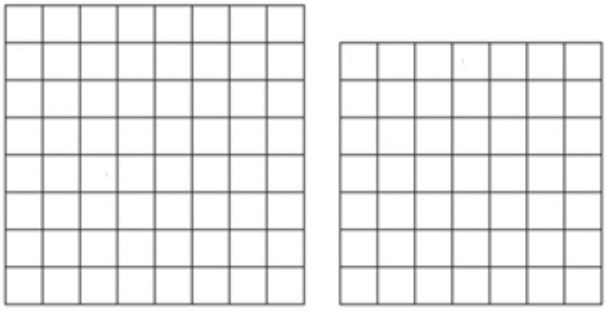
b) [*14 marks*] Device a divide and conquer algorithm to place $2n$ dots on n x n board, with the condition that no more than 2 dots are in the same row, column, or diagonal. Below is the solution for n = 8 (left), n = 7 (right).

When n = 8, board is 8 x 8, dots possible = 16 (i.e. $2n$)

When n = 7, board is 7 x 7, dots possible = 14 (i.e. $2n$)



The following grids are provided for your empty of dots to develop your solution;



You can use pseudo code or code. There is only one input; the empty n *x* n array A.

Solution

- *Observation*
  - Since $2n$ dots need to be placed into n rows and n columns for the board with at most 2 in the same row, or in the same column, then, exactly two dots have to be placed in each row and column.
- *Algorithm*
  - For even n=2k, a solution can be obtained by identical placement of n dots in the first k columns and the last k columns as follows (assuming rows and columns are numbered top to bottom and left to right):
    - place two dots in the first 2 rows of columns 1 and k+1,

- place two dots in rows 3 and 4 of columns 2 and k +2, and so on until finally dots are placed in rows n-1 and n of columns k and 2k.

- For odd $n = 2k + 1$, $k > 0$, a solution can be obtained by
  - place two dots in rows 1 and 2 of column 1,
  - place two dots in rows 3 and 4 of column 2, and so on until dots are placed in rows n-2 and n-1 of column k.
  - Then two dots are placed in the first and last rows of column k+1.
  - After that, k dots are placed in right part of the board symmetrically with respect to the board's central square to those in the left part:
    - two dots are placed in rows 2 and 3 of column k+2,
    - two dots are placed in rows 4 and 5 of column k+3, and so on, until rows n-1 and n of the last column.

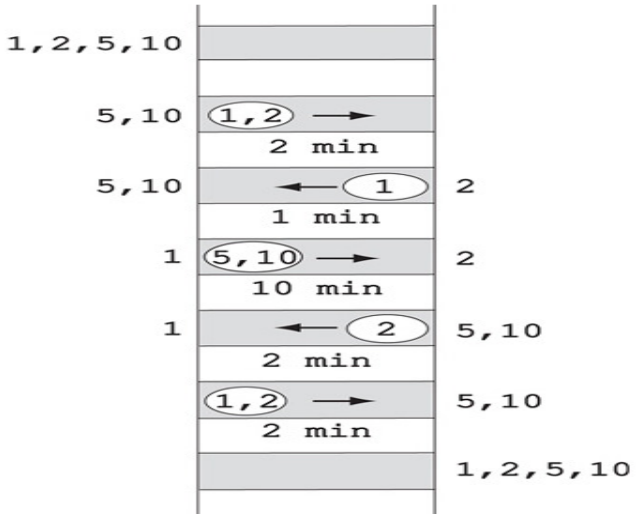
Question 2. Greedy Algorithm Design [*20 marks total*]

Imagine you are working in the command and control center of the Army's special operation units! Four soldiers get stranded in a hostile mountainous site. You are responsible for their safety. They need to path through a very narrow valley between 2 mountains. It is dark, and they have one battery. A maximum of two soldiers can cross the valley at one time. Any party that crosses, either one or two soldiers, must have the battery with them. The battery must be walked back and forth. It cannot be thrown, for example. Due to the payload they are carrying, it takes each soldier different amounts of time to cross. Solider 1 takes 1 minute, to cross the valley, solider 2 takes 2 minutes, solider 3 takes 5 minutes and solider 4 takes 10 minutes. A pair must walk together at the rate of the slower soldier's pace. For example if solider 1 and solider 4 walk together, it will take them 10 minutes to get to the other side. If solider 4 return the battery, a total of 20 minutes have passed. Design a greedy algorithm to cross the valley in the least amount of time. Concise English description of your algorithm is acceptable.

Solution

- *Observations*
  - The fastest solider returning back the battery is indeed a greedy approach to minimize returning back time. However, a better one is to minimize both forward and backward time. So, minimize the difference between the pair travelling together forward and also pick fastest going back from those arrived on other side. Thus, we need to pick the fastest to go on to the other side as early as possible to be able to use him in going back
- *Algorithm*
  - *Input configuration*: creates pairs with the heuristic to minimize the difference between the 2 parties involved in the pair (because that results in the biggest saving)

- *Objective function*: from the input configuration, pick the pair to cross forward first whose has the smallest value (to ensure fastest cross first, so we can use him in the return trip)
- In our example, the pairs are (5,10) (1,2)



Marking

- Each error or omission in a correct solution should be penalized up to 3 points

Question 3.Dynamic Programming Design [*20 marks total*]

a) [*8 marks*] A sequence of numbers $X[1..k]$ is exponentially-increasing if

$$X[i] > (2 \textit{ multiplied by } X[i-1]) \qquad \text{for all } 1 < i <= k.$$

For example; 2, 5, 11, 30 is exponentially increasing. Your task is to design a dynamic programming algorithm to find the largest exponentially-increasing subsequence in an input sequence $A[1\ldots n]$. Here is an example of an exponentially-increasing subsequence (the subsequence marked below):

1 10 3 5 11 7 25 53 12

Note that the subsequence members (1, 3, 11, 25, 53) could be separated by one or more values that do not belong to the subsequence. You can use either a top-down or a bottom-up approach in your solution. You can use pseudo code or code. There is only one input; the array A and the output is the subsequence.

Solution

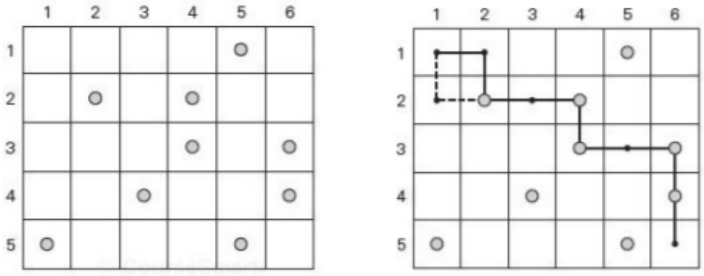
- *Observation*
  - Let $s(i)$ be a subsequence of length $S(i)$ that ends with $A[i]$. There are two possibilities,
    - (i) $S(i) = 1$, or
    - (ii) the second last element in $s(i)$ is a $j < i$ such that $A[i] > 2*A[j]$.
- *Algorithm*
  - We consider these possibilities to design our recursion.

$$S(i) = \max\left(1, \max_{j<i,\ A[i]>2A[j]} (S(j)+1)\right)$$

  - Let S be a table, we fill it iteratively similar to the maximum increasing subsequence problem. In the end, we return the maximum of all $S[i]$'s.

```
1: procedure MEIS(A[1 · · n])
2:     for i ← 1 to n do
3:         S[i] ← 1
4:         for j ← 1 to i − 1 do
5:             if A[i] > 2A[j] and S[i] < S[j] + 1 then
6:                 S[i] ← S[j] + 1
7:     return max_{0≤i≤n} (S[i])
```

b) [*12 marks*] Suppose some bread crumbs (☺ فتـــات الخـــبز) are spread in the cells of an *n* x *m* board, one bread crumb per cell. An ant discovers this food treasure and it is currently standing in the upper left cell of the board. The ant needs to collect as many of the bread crumbs as possible and bring them to the bottom right cell. On each step, the ant can move either one cell to the right or one cell down from its current location. When the ant visits a cell and finds that it has a bread crumb, it picks that. Devise a dynamic programming algorithm to find the maximum number of bread crumbs the ant can collect and a path it needs to follow to do that. For example, the solution to the grid with bread crumbs (represented as dots) on the left figure below is shown in the right figure. You can use either a top-down or a bottom-up approach in your solution. You can use pseudo code or code. The input is a 2D array *A* with empty cells set to null. The output is an array in which you have stored the cells to be visited.



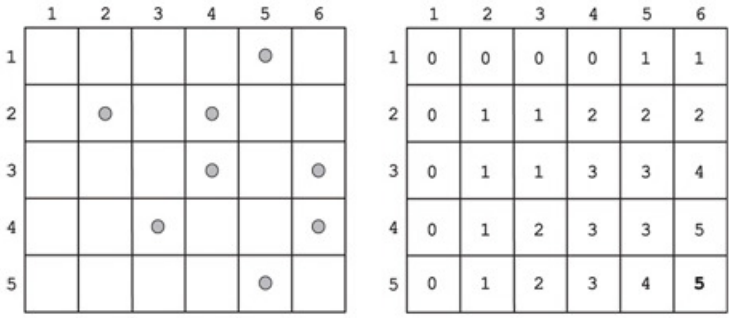
Solution

- Algorithm:
  - Let C[*i,j*] be the largest number of bread crumbs the ant can collect and bring to the cell (*i,j*) in the *i*th row and *j*th column of the board.
  - It can reach this cell either from the adjacent cell (*i*-1, *j*) above it or from the adjacent cell (*i,j*-1) to the left of it.
  - The largest number of breadcrumbs that can be brought to these cells are C[*i*-1,*j*] and C[*i,j*-1] respectively
  - Since there are no adjacent cells above cells in first row, and there are no adjacent cells to the left of cells in the first column. For such cells, we assume that C[*i*-1,*j*] and C[*i,j*-1] are zero.

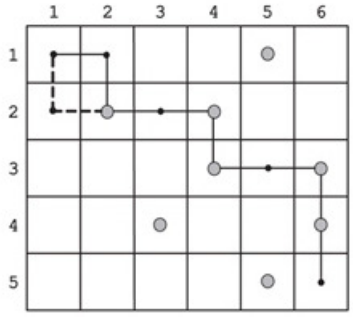
- Therefore the largest number of bread crumbs the ant can bring to cell (*i,j*) is the maximum of these two numbers plus one possible breadcrumb at cell (*i,j*) itself.
- The recursive formula

$$C[i, j] = \max\{C[i-1, j],\ C[i, j-1]\} + c_{ij} \text{ for } 1 \le i \le n,\ 1 \le j \le m,$$

where c_{ij} = 1 if there is a breadcrumb in cell (*i,j*) and c_{ij}=0 otherwise. and C[0,*j*] = 0 for 1 <= *j* <= *m* and C[*i*,0]=0 for 1 <= *i* <= *n*.



	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	0	1	2	3	4	**5**



Question 4.Asymptotic Analysis [*10 marks total*]

State if the following is true/false. **Justify your answer.**

a) [*5 marks*] $2^{3^n} \in \Theta(2^{3^{n+1}})$

T F

Solution

False. $2^{3^{n+1}} = 2^{3 \times 3^n} = (2^3)^{3^n} = 8^{3^n}$ which is not in $O(2^{3^n})$

This can also be verified by taking limits as $n \to \infty$

Marking

- *2 marks* for **<u>F</u>**.
- *3 marks* for justification.
  - Justification should be classified to: complete justification 3/3 vs. Incomplete Justification 1.5/3

b) [*5 marks*] $n^2 \in O(\frac{n^3}{\lg n})$

T F

Solution

True. Taking the limit of $\frac{n^2}{(\frac{n^3}{\lg n})}$ as $n \to \infty$, we get $\frac{\lg n}{n}$, which converges to 0.

Therefore, $n^2 \in O(\frac{n^3}{\lg n})$

Marking

- *2 marks* for **<u>T</u>**.
- *3 marks* for justification.
  - Justification should be classified to: complete justification 3/3 vs. Incomplete Justification 1.5/3

End of Midterm