**LAST NAME , FIRST NAME**

CS 4520/6520 Spring 2020

**Homework #2**

**Problem 1 [25 points]** *Review and algorithm design*

Consider the following problem:

**Given**: an unsorted array A of n−1 distinct integers from the range 1 to n.

**Find**: the missing integer.

Example: A = [7, 5, 1, 3, 4, 6]. The answer: 2 is missing

Propose 3 different algorithms to solve this problems with running times O(n), O(nlog(n)) and O(n2), respectively.

**1)**

**O(n)**

**x = (1+n)\*n/2**

**a = sum of A**

**ans = x- a**

**2)**

**O(nlogn)**

**Using merge sort to sort the array and linear search the missing num**

**3)**

**Brute force**

**Problem 2. [45pts].**

Sort the following array, showing each change **in bold blue font**.

60 13 11 77 66 67 21 27 14 71

1. By insertion sort method

13 60 11 77 66 67 21 27 14 71

11 13 60 77 66 67 21 27 14 71

11 13 60 77 66 67 21 27 14 71

11 13 60 66 77 67 21 27 14 71

11 13 60 66 67 77 21 27 14 71

11 13 21 60 66 67 77 27 14 71

11 13 21 27 60 66 67 77 14 71

11 13 14 21 27 60 66 67 77 71

11 13 14 21 27 60 66 67 71 77

1. By merge sort

60 13 11 77 66 || 67 21 27 14 71

60 13 || 11 77 66 || 67 21 || 27 14 71

60 || 13 || 11 ||77 66 || 67 || 21 || 27 || 14 71

60 || 13 || 11 ||77 || 66 || 67 || 21 || 27 || 14 || 71

13 60 || 11 || 77|| 66 ||21 67||27||14||71

11 13 60 || 66 77||21 27 67||14 71

11 13 60 66 77||14 21 27 67 71

11 13 14 21 27 60 66 67 71 77

1. By Quicksort

Choose the pivot to be the last element, i.e. 71.   
(Hint: you can follow the demo we watched and follow the procedure)

60 13 11 66 67 77 21 27 14 71. P = 71

60 13 11 66 67 21 77 27 14 71

60 13 11 66 67 21 27 77 14 71

60 13 11 66 67 21 27 14 71 77

13 60 11 66 67 21 27 14 71 77 p =14

13 11 60 66 67 21 27 14 71 77

13 11 14 60 66 67 21 27 71 77

11 13 14 60 66 67 21 27 71 77 p =11

11 13 14 21 66 67 60 27 71 77 p =27

11 13 14 21 27 66 67 60 71 77

11 13 14 21 27 60 66 67 71 77 p =60

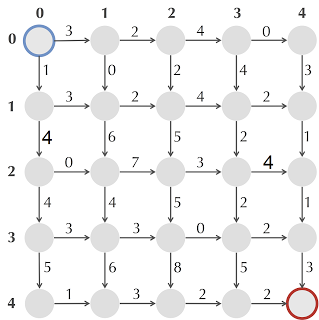
**Problem 3 [30pts].**

Imagine you’re a tourist on Manhattan, and that’s a map. You start at upper left corner (with coordinates 0,0) and should end up at the bottom right corner (with coordinates 4,4). Weights on edges indicate how many attractions you will see if you walk on that street/avenue.

Your goal is to see as many attractions as possible.

Fill in the matrices A (values, max numbers of attractions one can see up to that “road intersection”) and B (arrows, so one can reconstruct the path).

1. Using **greedy** approach



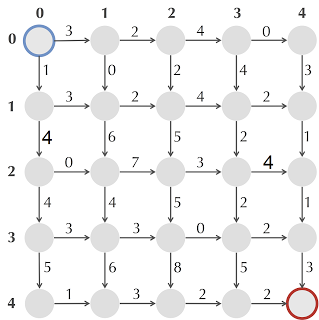
A:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 3 | 5 | 9 |  |
|  |  |  | 13 | 15 |
|  |  |  | 15 | 19 |
|  |  |  |  | 20 |
|  |  |  |  | 23 |

B: (copy-paste appropriate arrows) → ↓ → ←

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| → | → | → | → |  |
|  |  |  | ↓ |  |
|  |  |  | → | ↓ |
|  |  |  |  | ↓ |
|  |  |  |  | ↓ |

1. Using **Dynamic programming**



A:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 3 | 5 | 9 | 9 |
| 1 | 4 | 7 | 13 | 15 |
| 5 | 10 | 17 | 20 | 24 |
| 9 | 14 | 22 | 22 | 25 |
| 14 | 20 | 30 | 32 | 34 |

B: → ↓ → ←

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | ← | ← | ← | ← |
| ↑ | ← | ↑ | ↑ | ← |
| ↑ | ↑ | ← | ← | ← |
| ↑ | ↑ | ↑ | ↑ | ↑ |
| ↑ | ↑ | ↑ | ← | ← |