CSCI 3104 Exam2

Jonathan Phouminh

TOTAL POINTS

78 / 110

QUESTION 1

1 1a 1 / 1

✓ **+ 1 pts** **Correct**

+ 0 pts Incorrect. See solution

QUESTION 2

2 1b 1 / 1

✓ **+ 1 pts** **Correct**

+ 0 pts No or incorrect answer

QUESTION 3

3 2 6.5 / 7

+ 7 pts Correct

+ 0 pts No answer

+ 3 pts Some correct partitioning and uses first element of the subarray as the pivot, but the result is deviated significantly from the partition algorithm provided in Question 1. See solution.

+ 0 pts Incorrect.

✓ **+ 6.5 pts** **Minor errors**

+ 1 pts Provide the sub-lists. Without those, it is virtually impossible to follow your work

+ 2 pts A different partitioning algorithm used, and the resulting list is not sorted or missing an element.

+ 1 pts Correct partitioning, but uses last element of the subarray/sublist as the pivot element.

+ 5 pts Switched left and right children in the tree of recursive calls once, but uses the partitioning algorithm of Q1 correctly otherwise.

+ 2.5 pts Uses some correct partitioning different from the partition algorithm provided in Question 1. But has a minor error in the answer.

+ 3 pts Uses the partitoning algorithm of Question 1 for sorting correctly, but has made an error in the tree of recursive calls.

You are missing element 3 from your array, but I am not deducting points for that.

1 This subarray will be [3,1,5,2] after partitioning with 6 as pivot if we use the partitioning algorithm mentioned in Question 1.

QUESTION 4

4 3a 1.5 / 2

+ 2 pts Correct

+ 0 pts No answer

✓ **+ 1.5 pts** **Missing A[3], B[3] comparison**

+ 0.5 pts Significant errors

QUESTION 5

5 3b 3 / 3

✓ **+ 3 pts** **Correct example with explanation**

+ 0 pts No answer

+ 0 pts Significantly fewer comparisons than the maximum

+ 2.5 pts Your example requires 6 rather than 7 comparisons

+ 2 pts Correct example with incorrect or no explanation.

+ 1 pts Incorrect example.

QUESTION 6

6 3c 3 / 3

✓ **+ 3 pts** **Correct**

+ 1 pts Slightly too many comparisons

+ 0 pts The merge operation is in the context of Mergesort (as stated at the start of Problem 3). So A and B both need to be sorted with respect to the same order.

+ 0 pts No answer.

+ 2 pts Correct example with incorrect or no

explanation.

+ **1 pts** Incorrect example

QUESTION 7

7 4 **5 / 5**

✓ **+ 5 pts Correct**

+ **0 pts** Incorrect or not attempted.

+ **0 pts** The algorithm from class uses the end element as the pivot. Please adhere to the conventions from class.

+ **4 pts** Mostly correct partitioning, but returned wrong element

+ **4.5 pts** Correct partitioning, but did not answer the question. What is the kth smallest element here?

+ **1 pts** Significant errors in partitioning

+ **1.5 pts** Quick-select only examines one sub-array, not both

+ **4 pts** Some errors in partitioning

+ **3 pts** Missing details in partitioning after first step

QUESTION 8

8 5 **4.5 / 5**

✓ **+ 5 pts Correct**

+ **1 pts** Top-down only

+ **1 pts** Memoizes only

+ **3 pts** Top down with memoization, but does not utilize the memoization. If the desired element exists in the lookup table, you should use it.

+ **0 pts** Incorrect answer

- **1 pts** Missing base cases

- **0.5 pts** Swapped base cases

+ **0.5 pts** Handled base cases only

✓ **- 0.5 pts Incorrect base cases.**

+ **3 pts** Uses the top down technique with memoization, but does not set the computed value in the lookup table to use it in future.

+ **0 pts** (Non-scored rubric item) Recursive call function is being invoked with incorrect arguments.

+ **5 pts** Uses the top down with memoization technique correctly but initializes the lookup array inside the function. In this case, there is no advantage in using the memoization technique as the initialization causes the computed values to be overwritten.

- **1 pts** Minor errors in the algorithm

QUESTION 9

9 6a **3 / 3**

✓ **+ 3 pts Correct**

+ **0 pts** No or incorrect answer

+ **1 pts** Correct reasoning, but did not provide an example

+ **2 pts** You provided a graph but did not explain why this example worked.

QUESTION 10

10 6b **1.5 / 2**

+ **2 pts** Correct

+ **0 pts** No answer

+ **0.5 pts** Looks at previous two cases, but the recurrence is still not correct. Need to account for the weight of the current node if looking for at i-2 case. Also, you need to account for the base cases of when G is a single node or a path on two nodes.

+ **1.5 pts** Correct recursive step, but minor issues with base case

✓ **+ 1.5 pts Missing base cases**

+ **0 pts** Incorrect. See solution

+ **1 pts** Incorrect recursive case

QUESTION 11

11 6c **2.5 / 3**

+ **3 pts** Correct

+ **0 pts** No answer

+ **1 pts** Correct algorithm, but does not memoize

+ **2.5 pts** Otherwise correct, but need to consider case in which graph consists of a single vertex

+ **1 pts** Some correct logic, but significant errors

+ **0 pts** Incorrect. See solution

+ **2 pts** Some correct logic, but your algorithm generates index out of bounds errors

✓ **+ 2.5 pts Minor errors**

The optimal if the graph has 2 vertices is the maximum among the weight value of both these

vertices.

QUESTION 12

12 6d 0 / 3

+ 3 pts Correct

✓ + 0 pts No answer

+ 1.8 pts Two incorrect entries.

+ 2.4 pts One incorrect entry

+ 0 pts Incorrect

QUESTION 13

13 6e 0 / 3

+ 3 pts Correct

✓ + 0 pts No answer

+ 2.5 pts Clearly indicate optimal solution

+ 0 pts Incorrect answer

+ 2 pts Partially correct work

QUESTION 14

14 7a 0 / 2

+ 2 pts Correct

+ 1 pts Swapped order of C and B.

✓ + 0 pts Incorrect or no answer

+ 0 pts Use parentheses, and not curly braces, when denoting sequences. Sequences are denoted with parentheses, as order matters. Curly braces denote sets, in which order does not matter.

- 0.5 pts Listed strings not prescribed by the table.

+ 1 pts Missing first letter

QUESTION 15

15 7b 0 / 3

+ 3 pts Correct

✓ + 0 pts Incorrect or no answer. See solution.

+ 0 pts Incorrect. Modify the table, not the input strings

+ 1 pts Diagonal arrows can only point to the cell at the (i-1, j-1) index.

See solution

QUESTION 16

16 8a 0 / 2

+ 0.5 pts Correct cost of insertion

+ 0.5 pts Correct cost of deletion

+ 0.5 pts Correct cost of sub

+ 0.5 pts Correct cost of no-op

✓ + 0 pts No or incorrect answer

QUESTION 17

17 8b 2 / 2

✓ + 2 pts Correct

+ 0 pts No or incorrect answer. See solution

+ 0.5 pts Give entire strings, not just the end characters

+ 1.5 pts Minor errors

+ 1 pts Only one string is correct

Note that the rows are indexed by the x-coordinate, and the columns are indexed by the y-coordinate

QUESTION 18

18 8c 3 / 4

+ 4 pts Correct

+ 3 pts It is cheaper to sub the A in SCARE to the second U in CUUR.

+ 3 pts It is cheaper to insert S and leave the C unchanged, when converting CUUR to SCARE

✓ + 3 pts Nearly correct but a minor mistake.

+ 1 pts Correctly transforms, but is not minimum cost. See solution.

+ 0 pts No answer/Incorrect.

QUESTION 19

19 9a 4.5 / 5

+ 5 pts Correct

✓ + 4.5 pts Minor errors.

+ 4.5 pts Fill in all the cells

+ 2 pts Significant errors in constructing table. See solution

+ 0 pts No answer/Incorrect table

QUESTION 20

20 9b **5 / 5**

✓ **+ 5 pts Correct**

+ 2.5 pts One correct alignment

+ 4 pts Minor errors.

+ 0 pts No answer/Incorrect.

QUESTION 21

21 10 **3.5 / 8**

+ 8 pts Correct

+ 0 pts No answer

+ 1 pts Gave base case for example, but not general argument.

+ 2 pts Correct base case

+ 2 pts Recursive case 1: Result(i+1) if i <= n

✓ **+ 2 pts Recursive case 2: p(i) + Result(i + f[i] + 1) if i + f[i] + 1 <= n**

✓ **+ 1.5 pts Recursive case 2: Used i + f[i] rather than i + f[i] + 1**

+ 1.5 pts Recursive case 2: p(i) + Result(i + f[i] + 1), but you specified that i <= n rather than i + f[i] + 1 <= n. Be careful as to not go out of bounds

+ 1 pts Recursive case 2: p(i) + Result(i + f[i]), but you specified that i <= n rather than i + f[i] + 1 <= n. Be careful as to not go out of bounds

+ 2 pts Recursive case 3: p(i), otherwise.

+ 1 pts Recursive case 3: Used p(i+1) rather than p(i)

+ 0 pts Incorrect. See solution

+ 0 pts You are not being asked for a runtime complexity recurrence.

- 0.5 pts Need to specify conditions on indices for each recursive case to avoid going out of bounds

- 0.5 pts You are considering the array in reverse order. Note that Result(i) considers p[i, ..., n], and not p[1, ..., i].

✓ **+ 0 pts You were asked for a recurrence relation, not an algorithm.**

Base case is Result(n) = p[n]

QUESTION 22

22 11a **3 / 5**

+ 2 pts Correct capacity

✓ **+ 3 pts Correct edges**

+ 2 pts Listed only two correct edges, or added an extra edge.

+ 1 pts Only one correct edge in cut

+ 2 pts Correct cut, but did not list edges.

+ 0 pts Incorrect capacity and cut edges. See solution

- 0.5 pts Listed extraneous edge.

+ 0 pts No answer

QUESTION 23

23 11b **2 / 2**

✓ **+ 2 pts Correct**

+ 0 pts Incorrect. Current flow != capacity of min cut.

+ 1 pts Right answer, wrong reason.

+ 0.5 pts Right answer given wrong answer for 11a

+ 0.5 pts Correct flow, wrong answer.

+ 0 pts No answer

QUESTION 24

24 11c **3 / 3**

✓ **+ 3 pts Correct**

+ 0 pts Incorrect

+ 1 pts Right answer, wrong reason

+ 0.5 pts Right answer given wrong answer for 11a

+ 0 pts No answer

QUESTION 25

25 12 **3 / 8**

+ 8 pts Correct

+ 0 pts No answer

+ 4 pts Correct algorithm

+ 3.5 pts Minor errors in implementation

✓ **+ 1.5 pts Recognized to modify binary search and made some progress towards designing an algorithm, but did not successfully design an algorithm to do so**

+ 1 pts Correct base case, but incorrect searching

✓ **+ 1.5 pts Searches correct half of the array, but incorrect base case**

+ 4 pts Reasonable attempt at modifying binary search and runs in O(\log(n)) time.

+ **0 pts** Incorrect to check if L[mid] = k+1. There is no guarantee that if L[mid] = k+1, that L[mid-1] = key. Also, we don't know that the first element greater than key is key+1.

+ **0 pts** If A[mid] < k+1, you need to look right, rather than left.

+ **0 pts** If A[mid] > key, then mid-1 should be the new upper bound.

+ **0 pts** Does not run in O(log(n)) time

+ **0 pts** Incorrect algorithm

+ **0 pts** Searching for value greater than key linearly after finding key from binary search is not O(logn) in worst case

+ **0 pts** returns index of the key. Not what is being asked for.

QUESTION 26

26 13a 3 / 3

✓ **+ 3 pts Correct**

+ **2.5 pts** Missing case. When considering the (i, sum-results[k]) entry in the table, you need to ensure that sum - results[k] >= 0 first.

+ **0 pts** No or incorrect answer

+ **0.5 pts** Only base cases correct

- **0.5 pts** Missing base case or incorrect base case

- **0.5 pts** Missing bounds on second if statement

+ **1 pts** Right direction but major errors

+ **1.5 pts** Right direction, but your answer is not sufficiently precise that someone could reasonably implement an algorithm to fill in the lookup table

QUESTION 27

27 13b 3.5 / 4

+ **4 pts** Correct

+ **0 pts** No answer

+ **0 pts** Incorrect table

✓ **+ 3.5 pts Little error in table**

+ **3 pts** Error in table (needs to be a 5 x 5 table)

QUESTION 28

28 13c 0 / 3

+ **3 pts** Correct

✓ **+ 0 pts No answer**

+ **1.5 pts** Correct set, but incorrect arrows. Follow that T cells.

+ **0 pts** Incorrect arrows and no final set indicated

+ **1.5 pts** Correct arrows, but no or incorrect final set listed

+ **0.5 pts** Missing diagonal arrows for including elements. No final set included

QUESTION 29

29 14ab 8 / 8

✓ **+ 5 pts 14a- Correct**

+ **4 pts** 14a- Minor errors

+ **0 pts** 14a- Incorrect.

✓ **+ 3 pts 14b- Correct**

+ **1 pts** 14b- Some correct arrows, but still missing or incorrect arrows.

+ **0 pts** 14b- Incorrect

+ **0 pts** No answer

QUESTION 30

30 14c 2 / 2

✓ **+ 2 pts Correct**

+ **0 pts** No answer

+ **0 pts** Exceeds weight capacity

+ **1.5 pts** Minor errors

+ **0 pts** Incorrect.

Name: Jonathan Phaminh

ID: 106054641

CSCI 3104, Algorithms
Exam 2

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

Instructions: Please print your name and ID clearly in the boxes above. You are permitted a single sheet of notes, which cannot be shared with other students. Use the back of each page for scratch paper if you need it. This exam is otherwise a closed book, closed note, and an individual effort. Electronic devices are not allowed on your person (including in your pocket). Possession of such electronics is grounds to receive a 0 on this exam. Proofs should be written in **complete sentences. Show all work to receive full credit on questions where we have asked you to show your work.**

The exam has 110 possible points. You will be graded out of 100 points. You can choose to ignore 10 points worth of questions and still get a perfect score. If you attempt everything, you will still be graded out of 100, which can be beneficial if you have the time.

Circle your TA name:
Zhiyuan Liu
Wanshan Yang
Michael Levet
Connor Brooks

Please provide these:
Left neighbor name : Alex Palo
Right neighbor name : BAO NGUYEN

We provide the Master Theorem for your reference.

Master Theorem: Suppose $T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$.

(a) If there exists $c < \log_b(a)$ such that $f(n) \in \Theta(n^c)$, then $T(n) \in \Theta(n^{\log_b(a)})$.

(b) If $f(n) \in \Theta(n^{\log_b(a)})$, then $T(n) \in \Theta(n^{\log_b(a)} \log(n))$.

(c) If $f(n) \in \Theta(n^c)$, where $c > \log_b(a)$, then $T(n) \in \Theta(f(n))$.

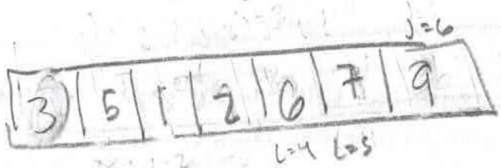
CSCI 3104, Algorithms
Exam 2

Name: JONATHAN PHOUMINH
ID: 106054641
Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

1. (2 pts) Use the array $A = [6, 1, 5, 9, 2, 7, 3]$ and this partition algorithm for the following questions. Read the partition algorithm closely.

```
Partition(A, lo, hi):
    pivot = A[lo]
    i = lo + 1
    for j = lo+1 to hi:
        if A[j] < pivot:
            swap(A[i], A[j])
            i = i + 1
    swap(A[i-1], A[lo])
    return i
```

← first index is partition



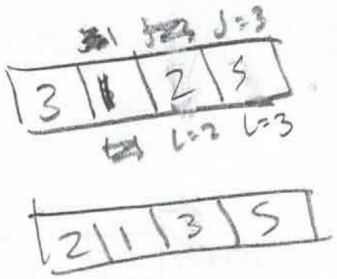
(a) (1 pt) What is the value of the pivot in the call $partition(A, 0, 6)$?

6

(b) (1 pt) What is the index of that pivot value at the end of that call to $partition()$?

A[4]

2. (7 pts) Assume Quicksort uses the **partition algorithm in Question 1** to sort the list $A = [6, 1, 5, 9, 2, 7, 3]$ in ascending order. Draw the tree of recursive calls that Quicksort makes, including the sub-lists and the indices that are passed to each call to Quicksort.

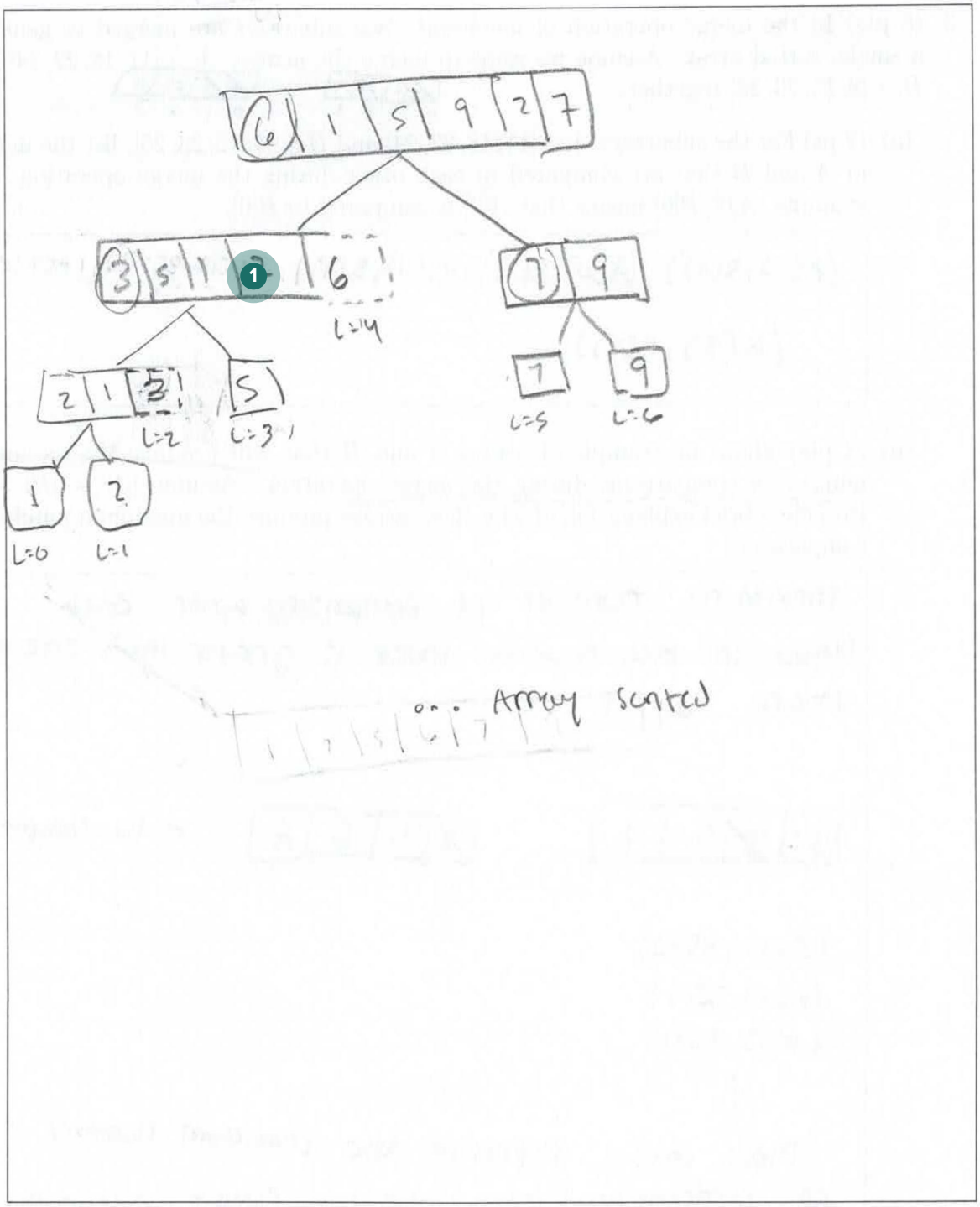


Name: Jonathan Phamanh

ID: 106054641

CSCI 3104, Algorithms
Exam 2

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder



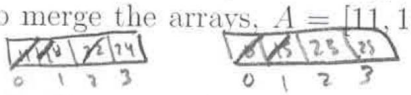
Name:

ID:

CSCI 3104, Algorithms
Exam 2

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

3. (8 pts) In the merge operation of mergesort, two subarrays are merged to generate a single, sorted array. Assume we want to merge the arrays, $A = [11, 18, 22, 24]$ and $B = [8, 15, 23, 25]$ together.



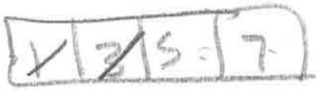
   (a) (2 pt) For the subarrays $A = [11, 18, 22, 24]$ and $B = [8, 15, 23, 25]$, list the indices in A and B that are compared to each other during the merge operation. For example, $A[0], B[0]$ means that $A[0]$ is compared to $B[0]$.

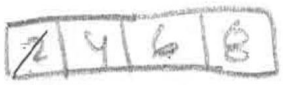
$(A[0], B[0])$, $(A[0], B[1])$, $(A[1], B[1])$, $(A[1], B[2])$, $(A[2], B[2])$

$(A[3], B[2])$

   (b) (3 pts) Show an example of arrays A and B that will produce the maximum number of comparisons during the merge operation. Assume $|A| = |B| = 4$. Provide a brief explanation of why these arrays produce the maximum number of comparisons.

maximum number of comparisons when each number in each relative index is greater than one of them by 1





~10 comparisons

$(A[1], A[2])$
$(A[2], B[1])$
$(A[1], B[2])$
⋮

These arrays represent the maximum number of comparisons because we will compare essentially all values with eachother except for one

(c) (3 pts) Show an example of arrays A and B that will produce the minimum number of comparisons during the merge operation. Assume the length of $|A| = |B| = 4$. Provide a brief explanation of why these arrays produce the minimum number of comparisons.

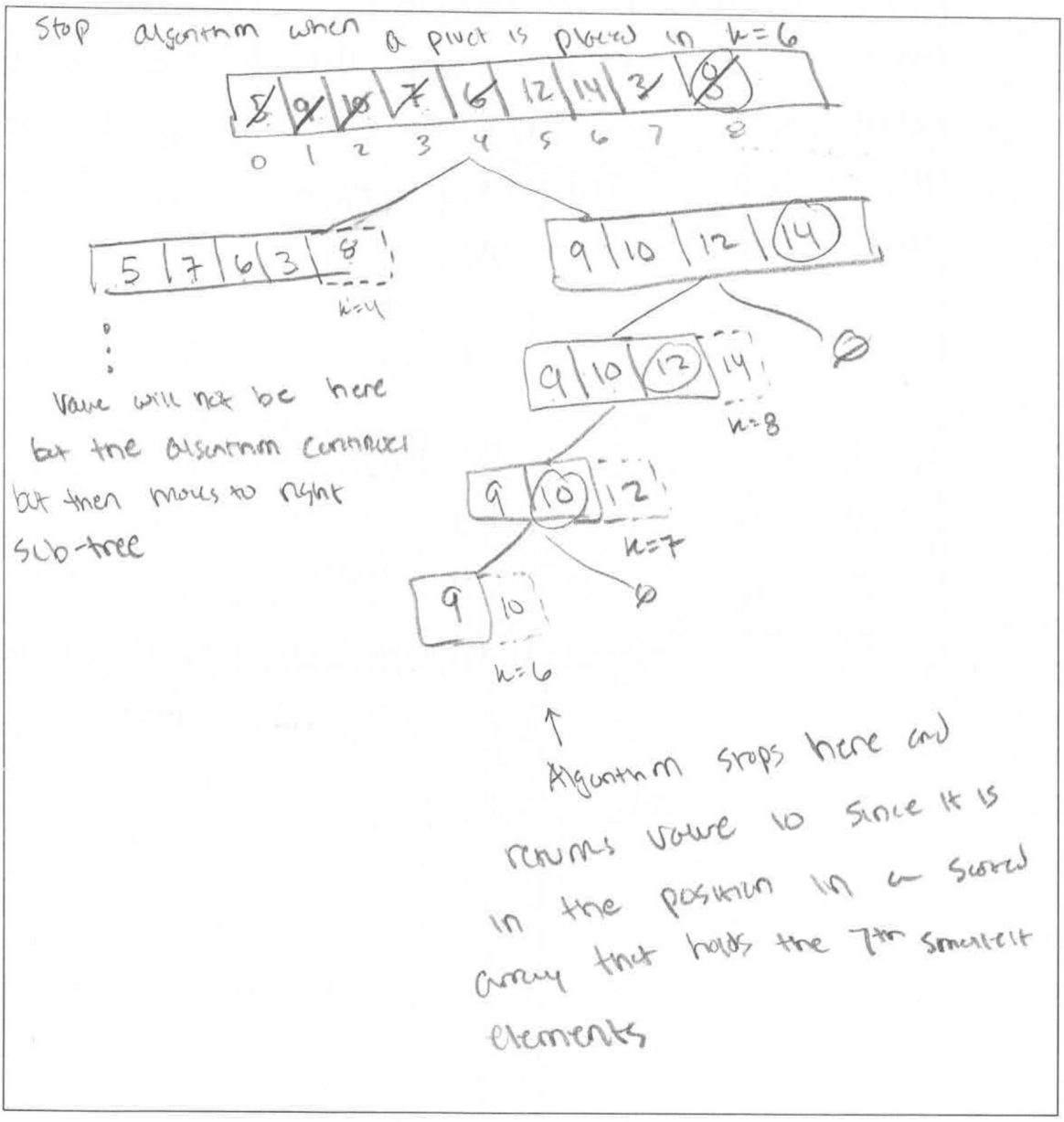
minimum number of comparisons when one item in either array is greater than all items in the other array and the item that is greater than all values in the other array happens to be the minimum of its own array

ex:

A = | 1 | 1 | 1 | 1 | B = | 2 | 3 | 4 | 5 | ~4 comparisons

(A[0], B[0])
(A[1], B[0])
(A[2], B[0])
(A[3], B[0]) ... Array A is exhausted so no more comparisons...

4. (5 pts) Illustrate how to apply the QuickSelect algorithm to find the $k = 7$th smallest element in the given array: `A = [5, 9, 10, 7, 6, 12, 14, 3, 8]` by showing the recursion call tree. Your answer needs to show the pivot and partitioned subarray that is passed to each recursive call to receive credit. Use a partition algorithm that selects the last value in the subarray as the pivot value.



5. (5 pts) The Lucas numbers are an integer sequence with the same recurrence relation as the Fibonacci numbers, but with a different base case.

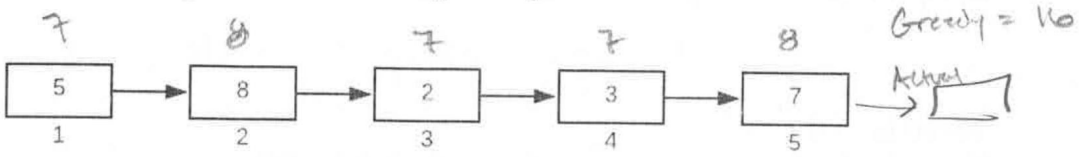
$$L_n = L_{n-1} + L_{n-2} \tag{1}$$

where $L_0 = 2$ and $L_1 = 1$.
Write a top-down recursive algorithm with memoization to calculate the first n Lucas numbers.

```
LucasNumbers(memo, n):
    if memo[n] != None:
        return memo[n]
    if n == 2:
        memo[n] = 2
        return memo[n]
    if n == 1
        memo[n] = 1
        return memo[n]
    else
        memo[n] = LucasNumbers(memo, n-1) +
                  LucasNumbers(memo, n-2)

    return memo[n]
```

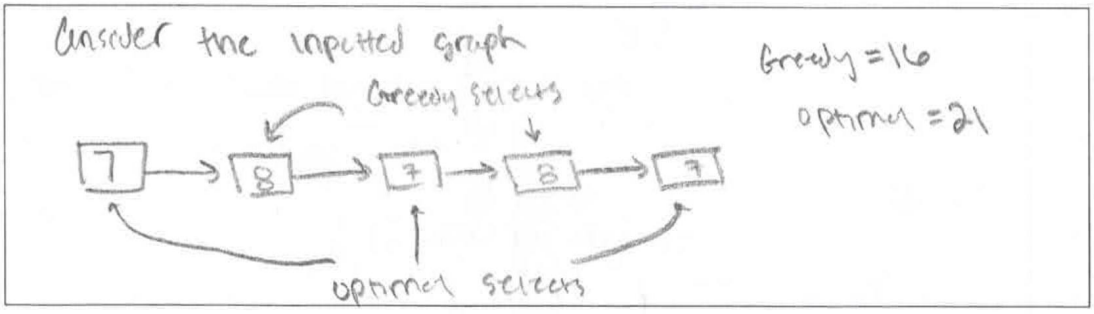
6. (14 pts) Consider the following graph $G = (V, E)$ with n vertices, where each vertex has at most one in-degree an one out-degree edge.

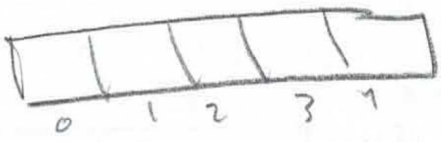


Each vertex v_i, has a positive integer weight w_i. For example, v_2 has a weight, $w_2 = 8$ An independent set on this graph is a set S of vertices such that for all distinct vertices $i, j \in S$, i and j are not connected by a single edge. A maximum independent set is the independent set where the sum of the vertex weights is maximum.

(a) (3 pts) Consider the following greedy algorithm for finding the maximum independent set. Provide an example where the greedy algorithm doesn't produce the optimal solution.

```
Start with S equal to the empty set
While some node remains in G
    Pick a node v_i of maximum weight
    Add v_i to S
    Delete v_i and its neighbors from G
Endwhile
Return S
```





Name: Jonathan Phaminh
ID: 1060546041

(b) (2 pts) Write the recurrence relation for a dynamic programming approach that produces the optimal solution.

knapsack?

$$opt[i] = \max \begin{cases} V_{iw} + S[i-2] & \text{// current node + previous solution} \\ S[i-1] & \text{// previous solution} \end{cases}$$

(c) (3 pts) Write either a bottom-up or top-down algorithm with memoization that implements your recurrence relation.

```
#n is the graph input
bottom_find max(n):
    optimal = []
    temp = []
    firstnode = n[0]
    secondnode = n[1]
    temp.append(firstnode)
    temp.append(secondnode)
    for i in range(2,n):
        optimal.append(max(temp[i-1], temp[i-2]) + n[i]
        temp.append(optimal[i-2])
    size = len(optimal)
    return optimal[size]
```

use 2 arrays, one for optimal solution, one for finding the actual solution

put optimal value in other array

Find the optimal

(d) (3 pts) For the graph G shown above, show how your algorithm completes a dynamic programming table by providing the filled table.

(e) (3 pts) Produce the optimal solution (maximum independent set) for graph G by showing the backtracking process through parent arrows to recover the solution from your table and also provide the vertices that give us the optimal solution.

7. (5 pts) Let $\omega = (A, B, C, D)$, and let $\tau = (B, A, C, B, D)$. Suppose we wish to find a longest common subsequence of ω and τ.

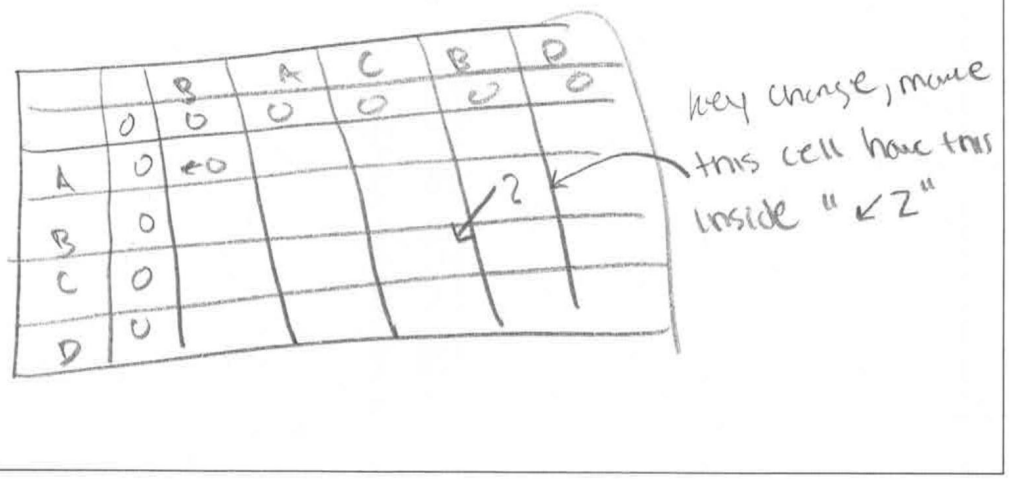
   (a) (2 pts) Consider the following table constructed from our dynamic programming algorithm. What is the longest common subsequence **prescribed by this table?**

		B	A	C	B	D
	0	0	0	0	0	0
A	0	$\leftarrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$
B	0	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 2$	$\leftarrow 2$
C	0	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\leftarrow 2$
D	0	$\uparrow 1$	$\leftarrow 1$	$\uparrow 2$	$\leftarrow 2$	$\nwarrow 3$

ABD

ABD

   (b) (3 pts) Modify the lookup table in such a way that (A, C, D) is the prescribed longest common subsequence. You can re-draw the table and just fill the cells which are modified **OR** you can just indicate the cells (example - table[1][1] $= \leftarrow 0$) assuming that the rows and columns are $0-$*indexed.*



8. (8 pts) Consider the source sequence `s = CUUR`, the target sequence `t = SCARE`, and the given complete cost table when you align `s` and `t` optimally.

	s\t	-	S	C	A	R	E
0	-	0	2	4	6	8	10
1	C	1	2	2	4	6	8
2	U	2	3	3	4	6	8
3	U	3	4	4	5	6	8
4	R	4	5	5	6	5	7

(a) (2 pts) What are the costs cost(insert), cost(delete), cost(sub), cost(no-op) used to produce this table.

insert: 1 | delete | sub: 111

one insert and three subs

(b) (2 pts) At the position (2, 4) in the cost matrix, what are the two substrings that are being aligned (assuming that the first row and column has index 0)?

SC
CUUR

(c) (4 pts) Using the given cost table, draw the optimal alignment that transforms `s = ØUUR` to the target sequence `t = ØCARE`. An example drawn alignment for a different example -

Drawn alignment = **Ops =** `['sub', 'sub', 'no-op', 'delete']`

S T E P
| | : |
A P E _

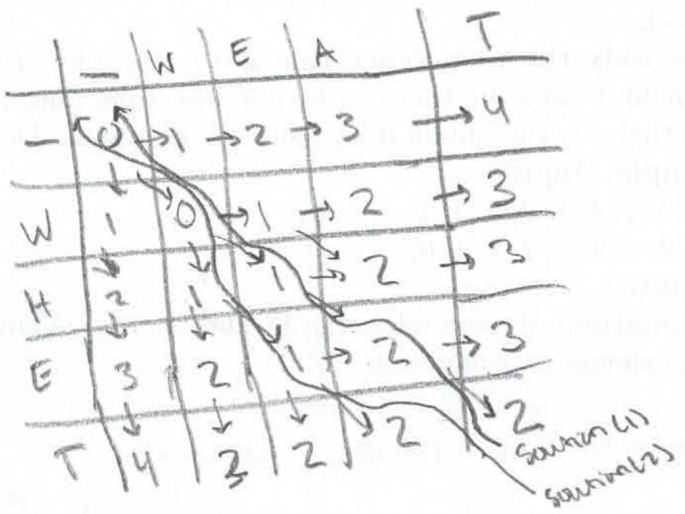
[sub, sub, sub, no-op, insert]

SCARE
CURR

S C A R E
| : | | :
C U R R —

9. (10 pts) Consider the problem of sequence alignment and the source sequence `s = WEAT` and the target sequence `t = WHET`. S = WEAT t = WHET

   (a) (5 pts) Create and fill in the DP table values using cost(insert) = 1, cost(delete) = 1, cost(sub) = 1, cost(no-op) = 0.



   (b) (5 pts) There are multiple optimal alignments of these strings. Draw and show at least two optimal alignments, including the parents in the backtracking. (If there is an overlap of parents, you can just draw one arrow for that step.)

Solution 1 = [r no-op, sub, sub, no-op]

W E A T
: | | :
W H E T

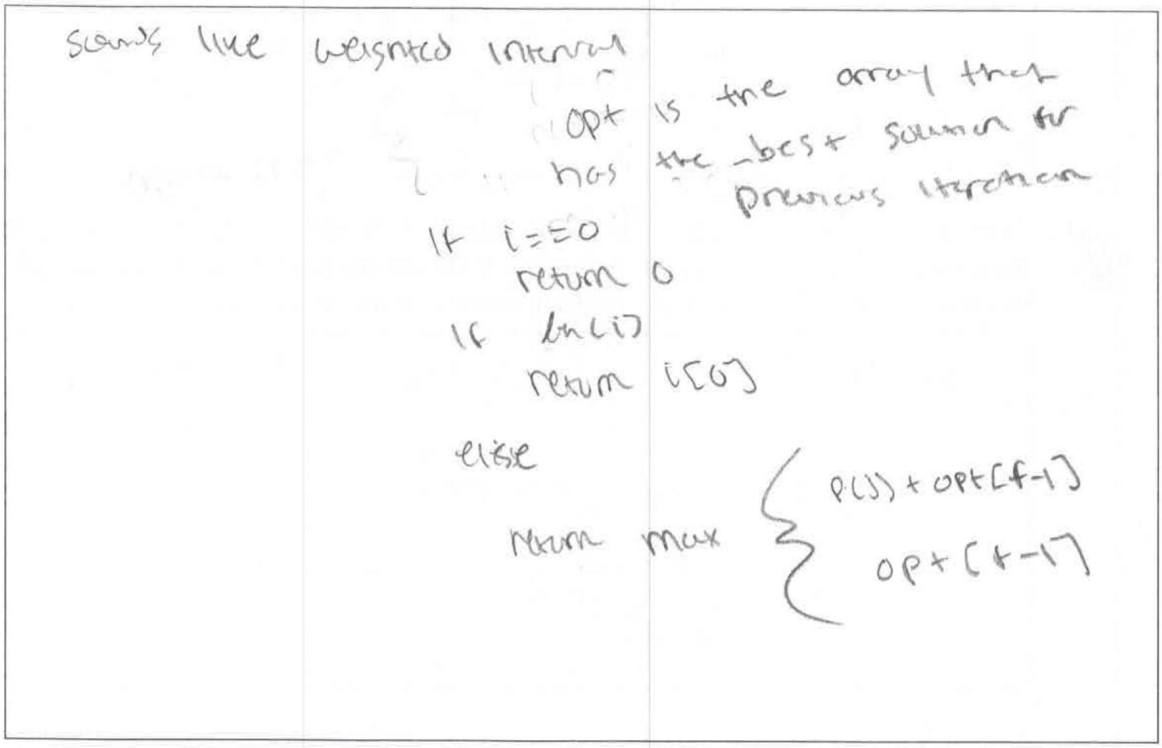
Solution 2 = [no-op, insert, no-op, sub, insert]

W E H T —
: : | | :
W — H E T

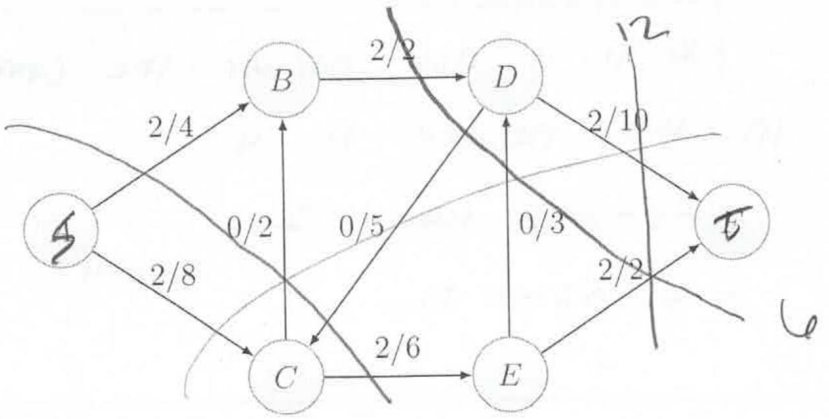
10. (8 pts) You are given an exam with questions numbered $1, 2, 3, ..., n$. Each question i is worth p_i points. You must answer the questions in order, but you may choose to skip some questions. The reason you choose to do this is that even though you can solve any individual question i and obtain the p_i points, some questions are so frustrating that after solving them you will be unable to solve any of the following f_i questions. Suppose that you are given the p_i and f_i values for all the questions as input. Suppose you want to develop an DP algorithm for choosing set of questions to answer that maximizes your total points. The first step is usually to come up with the recurrence relation.

Devise **only the recurrence relation** to solve for $Result(i)$ where $Result(i)$ is the maximum total score that can be obtained from questions i through n. It is not the score that you can obtain from the first i questions. Do not forget the base case.

Example: Input:
$p_i = [3, 2, 4, 1, 2, 5, 2, 1]$
$f_i = [2, 2, 3, 3, 2, 2, 1, 0]$
Output: 8
Explanation: We can select the 1^{st} and the 6^{th} assignment to get $3 + 5 = 8$ which is the maximum we can obtain.



11. (10 pts) Answer the following for the flow network with capacities as the denominator and the current flow as the numerator. A is the source and F is the sink.



(a) (5 pts) Find the minimum capacity cut and report the edges in this cut and the minimum capacity you found.

min cut = {F, D}, {A, B, C, E}

flow = 6

edges = (E, F), (E, D), (B, D)

(b) (2 pts) Is this minimum capacity equal to the current flow shown in the network? Justify your answer by providing the current net flow in the network.

NO it is not because the current flow in the network is 4

A → B → D → F flow is 2

A → C → E → F flow is 2

2+2=4

(c) (3 pts) Is this current flow the max flow? Justify in a sentence.

no the current flow is not the max flow, we know this because the max flow is equivalent to the min-curs flow cw in our case we dont have that because 4 ≠ 6

12. (8 pts) You are given a sorted list of numbers where duplicates are present. Write an algorithm that runs in $\mathcal{O}(\log(n))$ time and takes as input the list `L` and `key` and outputs the index of the first number that is greater than the `key`.

Example 1:

Input: L = {2,3,3,5,5,5,6}, key = 2
Output: 1
Explanation: The first 3 is the first number greater than 2 and has index 1

Example 2:

Input: L = {2,2,2,5,5,5,6}, key = 2
Output: 3
Explanation: The first 5 is the first number greater than 2 and has index 3

Implement Binary Search

```
BinarySearch(Array, p, r, key):
  if p<r
    mid = floor((p+r)/2)
    if Array[mid] == key:
        return Array[mid]
    elif Array[mid] > key:
        BinarySearch(Array, p, mid-1, key)
    else
        BinarySearch(Array, mid+1, p, key)
  else
    return -9999
```

13. (10 pts) Suppose we are given a set of non-negative distinct numbers s and a target t, and we want to find if there exists a subset of s that sums up to **exactly** t. [**Note:** This is different than Knapsack, where the elements in our subset can add up to be less than t.]
For example -

Input: s = {2,1,5,7}, t = 4
Output: False
Explanation: No subset sums to 4.

Input: s = {2,1,5,7}, t = 6
Output: True
Explanation: Subset {1, 5} sums up to the target $t = 6$.

Any sub-problem can be represented by `k` and `sum`, where, like Knapsack, `k` represents the first `k` numbers of the given set `s`, and `sum` represents any target value from $0 \leq$ `sum` $\leq$ `t`.

(There is a detailed example worked out on next page that shows the dynamic programming table that would be developed in a solution to this problem.)

Here are two worked out examples (corresponding to the above examples) with the DP table included were `T` corresponds to `True` and `F` corresponds to `False` which is filled using the recursive relation/solution.
The top example corresponds to `s = {2, 1, 5, 7}, and t = 4 (answer is False)` and the bottom example corresponds to `s = {2, 1, 5, 7}, and t = 6 (answer is True)`. If you look at the highlighted cell in the top example, it corresponds to the answer with the 1^{st} *three* elements of `s` and a target of `sum = 3` and the answer is `True`

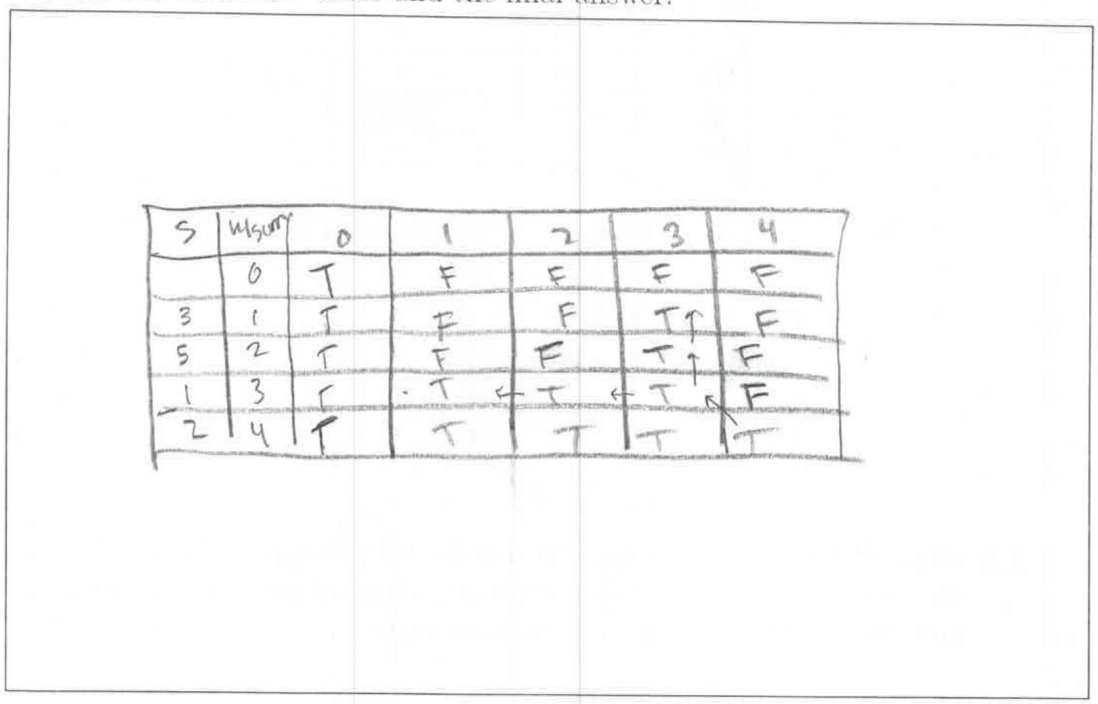
s	k \ sum	0	1	2	3	4
	0	T	F	F	F	F
2	1	T	F	T	F	F
1	2	T	T	T	T	F
5	3	T	T	T	T	F
7	4	T	T	T	T	F

s	k \ sum	0	1	2	3	4	5	6
	0	T	F	F	F	F	F	F
2	1	T	F	T	F	F	F	F
1	2	T	T	T	T	F	F	F
5	3	T	T	T	T	F	T	T
7	4	T	T	T	T	F	T	T

(a) (3 pts) Write the recurrence relation (or the recursive solution) for an instance of `(k, sum)` in terms of smaller relevant sub-problems. Do not forget to complete your recurrence relation with the base case.

If sum - S[k] == 0:
    TABLE[k-1, sum] || TABLE[k-1, sum - S[k]]
Otherwise: Table[k, sum] = TABLE[k-1, sum]
Base case
    Table[0,0] = TRUE
    And TABLE[0, sum > 0] = false

(b) (4 pts) Using the bottom-up approach solve for `s = {3, 5, 1, 2}, and t = 4`. Provide the entire DP table and the final answer.



S	w/sum	0	1	2	3	4
	0	T	F	F	F	F
3	1	T	F	F	T↑	F
5	2	T	F	F	T↑	F
1	3	T	T←	T←	T↑	F
2	4	T	T	T	T	T

(c) (3 pts) Draw the parent arrows for the sub-problems that are involved in the back-tracing of the final answer (You can draw it on the table you created in part(b)). Do not draw the parents which are not part of the final solution. Provide the subset that yields the answer in part(b).

Name: Jonathan Plumlee

ID: 106054641

CSCI 3104, Algorithms
Exam 2

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

14. (10 pts) Consider the DP table for the Knapsack problem with $W = 7$, and a list of items $A = [(3, 8), (5, 11), (3, 11), (4, 6), (2, 8)]$ of (weight, value) pairs.

(a) (5 pts) Fill in the remaining cells with the optimal values.

(b) (3 pts) Draw the arrows corresponding to the final optimal answer. You do not need to draw the parent pointers for the cells that are not part of the final optimal solution.

W = 7		k \ w	0	1	2	3	4	5	6	7
Weight	Value	0	0	0	0	0	0	0	0	0
3	8	1	0	0	0	8	8	8	8	8
5	11	2	0	0	0	8	8	11	11	11
3	11	3	0	0	0	11	11	11	19	19
4	6	4	0	0	0	11	11	11	19	19
2	8	5	0	0	8	11	11	20	20	20

(c) (2 pts) Provide the subset of items in the optimal solution.

optimal = {(2,8), (3,11)}