CSCI 3104 Final Exam

Jonathan Phouminh

TOTAL POINTS

68 / 100

QUESTION 1

10 pts

1.1 5 / 5

+ **2 pts** One of the three errors mentioned correctly

+ **4 pts** Two of the three errors mentioned correctly

✓ **+ 5 pts** **All three errors mentioned correctly**

+ **0 pts** Either incorrect or empty solution submitted

- **5 pts** Plagiarism

1.2 5 / 5

✓ **+ 5 pts** **Correct**

+ **0 pts** Incorrect

- **5 pts** Plagiarism

QUESTION 2

25 pts

2.1 5 / 5

✓ **+ 2 pts** **Gives correct example where BFS and Dijsktra's algorithm will not work.**

✓ **+ 1.5 pts** **Written valid argument why BFS will not work**

✓ **+ 1.5 pts** **Written valid explanation why Dijsktra's algorithm will not work.**

+ **0 pts** Incorrect answer or Not attempted

- **5 pts** Plagiarism

- **1 pts** Does not provide a clear example to prove the fact that Dijsktra and BFS may not work for finding minimum k-stop paths. But provides accurate justification for this.

- **1 pts** No answer for BFS or Dijkstra's

+ **1 pts** Explanation for why BFS/Dijsktra's fail is not completely correct

+ **2 pts** Writes valid explanation about why Dijsktra's and BFS fail but does not provide any example

2.2 1 / 10

+ **6 pts** (BFS/DFS approach): Algorithm computes paths of less than or equal to k-stops, from source to destination using BFS or DFS traversal.

+ **3 pts** (BFS/DFS approach): The algorithm correctly chooses a smallest k-stops path among all k-stop paths.

+ **3 pts** (Bellman-Ford approach): Identified the subproblems correctly i.e. path(i, j) = min(u(k, i) + path(i-1,k)) for k=1,...n

+ **6 pts** (Bellman-Ford approach): Algorithm finds the shortest k-stop path from source to destination using the previous recurrence relation.

+ **1 pts** Provides clear explanation of the algorithm

+ **4 pts** (3D approach): Identified the subproblems correctly i.e. path(i,j,k) = min(u(p, j) + path(i,p,k-1)) for all nodes adjacent to node j

+ **5 pts** (3D DP approach) Correctly finds the shortest k-stop path using the recurrence.

+ **0 pts** Incorrect answer or Not attempted.

- **10 pts** Plagiarism

+ **3 pts** Tries to use Modified DFS/BFS or Modified Bellman-Ford or 3D DP approach. But the algorithm does not find the correct k-stop minimum weighted path.

+ **0 pts** The algorithm just finds the shortest path from source to destination, not k-stops path.

+ **0 pts** The algorithm does not return the k-stop path but finds and returns only the weight of minimum k-stop path.

+ **3 pts** Uses Dijsktra's algorithm. The algorithm does not work for all inputs.

+ **3 pts** Algorithm does not explain all the steps clearly, but uses the correct approach.

+ 1 Point adjustment

2.3 0 / 10

+ **4 pts** (BFS/DFS approach): Identified that the k-stop paths can be found using BFS or DFS from source to destination. Traverses the graph and builds paths from source.

+ **2 pts** (BFS/DFS approach): Stops adding vertices to path after k hops

+ **3 pts** (BFS/DFS approach): Chooses a smallest k-stops path among all k-stop paths.

+ **3 pts** (Bellman-Ford DP approach): Identified the subproblems correctly i.e. path(i, j) = min(u(k, i) + path(i-1,k)) for k=1,...n

+ **4 pts** (Bellman-Ford DP approach): Correctly computed the table using the above recurrence relation

+ **2 pts** (Bellman-Ford DP approach) Correctly finds the shortest k-stop path from the table.

+ **1 pts** Handled the case where there is no path with k-stops from source to target

+ **3 pts** (3D DP approach): Identified the subproblems correctly i.e. path(i, j,k) = min(path(p, j,k-1) + path(i,p)) for all the edges (i,p)

+ **4 pts** (3D DP approach): Correctly computed the table using the above recurrence relation

+ **2 pts** (3D DP approach) Correctly finds the shortest k-stop path from the table.

- **1 pts** Implementation works correctly, but does not keep track of the path (sequence of vertices) from source to destination

+ **0 pts** Incorrect answer or Not attempted.

+ **0 pts** Implemented the DFS/BFS approach but does not keep track of weights of the path to find the minimum k-stop paths

- **10 pts** Plagiarism

+ **2 pts** Tries to use the right approach (Modified BFS/DFS or Modified Bellman-Ford or 3D DP), but the implementation has several bugs and gives incorrect result.

+ **0 pts** Code could not be executed.

- **5 pts** The code correctly computes the weight of minimum k-stop path. It does not find or incorrectly finds the path.

+ **4 pts** Uses Dijsktra's algorithm or a slight modification of Dijsktra's but the resulting code does not work for all inputs because of minor bugs

+ **4 pts** Tries to use the right approach (BFS/DFS or Bellman-Ford), but the implementation has minor bugs.

+ **2 pts** Uses shortest path algorithm with incorrect changes or without any changes to find the k-stop minimum weighted oath

✓ **+ 0 pts** **No code**

QUESTION 3

25 pts

3.1 10 / 10

✓ **+ 10 pts** **Correct**

+ **5 pts** The algorithm is correct which can always give a optimal solution.

+ **3 pts** The recurrence/transition function is correct.

+ **2 pts** The runtime is in Thetha-(n^2). PS: When doing recursion without memoization, the run time won't satisfy the requirement.

+ **3 pts** The algorithm is partial reasonable.

+ **0 pts** Incorrect/Not attempted.

- **10 pts** Plagiarism

3.2 15 / 15

✓ **+ 15 pts** **Correct**

+ **8 pts** The implementation corresponding to the transition function are totally correctly whatever recursive with memoization or DP. Here would be S[i][j] = min[C[i] + max(S[i+2][j],S[i+1][j-1]),C[j] + max(S[i+1][j-1],S[i][j-2])].

+ **2 pts** Deal with base case correctly when left with only two cards.

+ **4 pts** Have a function to randomly generate test data.

+ **1 pts** The result of output is well-organized and correct.

- **5 pts** If using recursive without memoization, the runtime won't be satisfied.

+ **3 pts** The algorithm implementation is partial correct.

+ **5 pts** The algorithm implementation is mostly correct but some minor mistakes.

+ **0 pts** Incorrect/Not attempted.

- **15 pts** Plagiarism

QUESTION 4

22 pts

4.1 8 / 8

✓ **+ 8 pts Correct**

+ **7 pts** Minor errors

+ **0 pts** Incorrect

✓ **+ 0 pts You can reduce your search space to A[0, ..., n/4] and A[3n/4, ..., n-1] right off the bat, just based on the problem statement**

+ **0 pts** No answer

- **4 pts** Not sub-linear time

+ **5.5 pts** How are you using findValley to get the two local mins?

- **8 pts** Plagiarism

+ **5.5 pts** Incorrect base cases

+ **5 pts** Significant errors

4.2 2 / 10

- **0 pts** Correct

- **3.5 pts** Need to explicitly cite correctness of findValley function

- **8 pts** This is not a proof

- **3 pts** Does not address how you are incorporating the findValley method to find the global minimum in the array.

✓ **- 8 pts Missing significant details**

- **0 pts** You can assume the correctness of binary search, findValley and findPeak. You did not have to attempt to prove them correct.

- **3.5 pts** You cannot assume A[0, .., n/4] and A[3n/4, ..., n-1] are sorted.

- **10 pts** No answer

- **3 pts** Inductive hypothesis always follows base case

- **2 pts** Missing base cases

- **10 pts** Incorrect

- **2 pts** Attempts to address how findValley is being utilized, but does not clearly articulate these details

- **4 pts** Incorrect structure for inductive proof. An inductive proof has the base case(s), the inductive hypothesis, and the inductive proof

✓ **- 0 pts A direct argument would suffice. "By assumption, the first local minimum is located in A[0, ..., n/4], and the second local minimum is located in A[3n/4, ..., n-1]. So it suffices to search these two sub-arrays. We apply the findValley algorithm to each of A[0, ..., n/4] and A[3n/4, ..., n-1]. Recall from class that the findValley correctly returns a local minimum in an array..." Now finish the proof**

- **10 pts** Plagiarism

- **3 pts** Incorrect claims or missing details

- **10 pts** Claimed proof of main algorithm was trivial, without spelling out the details

1 Your argument is circular. You are deriving a contradiction that the algorithm does not work, by assuming the algorithm works. This is not how proof by contradiction works.

4.3 4 / 4

✓ **+ 4 pts Correct**

+ **0 pts** Incorrect

+ **0 pts** No answer

+ **4 pts** Full credit for mostly correct work and correct final runtime complexity. Note that technically, your main algorithm is not recursing on itself. Your main algorithm invokes a separate recursive algorithm, whose runtime complexity we label A. So really T(n) = 2A(n/4) + O(1), where A(n) = A(n/2) + O(1).

+ **2 pts** Correct analysis of findValley or only.

+ **0 pts** Note that technically, your main algorithm is not recursing on itself. Your main algorithm invokes a separate recursive algorithm, whose runtime complexity we label A. So really T(n) = 2A(n/4) + O(1), where A(n) = A(n/2) + O(1).

+ **2 pts** Correct answer, but works is either unclear

or has some errors or not complete

- 4 pts Plagiarism

+ 2 pts Incomplete or Incorrect answer but started correctly with the correct equation.

+ 0 pts No supporting work

- 1 pts Minor errors

+ 1 pts Correct conclusion, but very incorrect work

QUESTION 5

5 13 / 18

✓ **+ 2 pts** **Identified that it is always possible to achieve sum zero using no elements from the set. or another appropriate base case.**

✓ **+ 2 pts** **Correct overall run time as part of the code comments**

+ 3 pts O(ntk) approach: Identified sub-problems are of the form answer[k][i][j] which represents, if it is possible to achieve sum j, using exactly k elements from first i elements

+ 7 pts O(ntk) approach: Realized that solution to answer[k][i][j] can be obtained from anwers to subproblems answer[k][i-1][j] and answer[k-1][i-1][j-s[i]]

+ 4 pts O(ntk) approach: Reconstruction of the actual solution by storing the parents.

✓ **+ 3 pts** **Subsetsum + Backtracking approach: Identified sub-problems are of the form answer[i][j] which represents, if it is possible to achieve sum j, using first i elements.**

✓ **+ 7 pts** **Subsetsum + Backtracking approach: Realized that answer[i][j] can be obtained from answers to subproblems answer[i-1][j] and answer[i-1][j-s[i]].**

+ 4 pts Subsetsum + Backtracking approach: Reconstruction of the actual solution by storing the parents.

+ 3.5 pts O(ntk) approach: Partially correct subproblems.

+ 3.5 pts Subsetsum + Backtracking approach: Partially correct subproblems.

+ 0 pts Incorrect

+ 0 pts Not attempted

- 18 pts Plagiarism

- 2 pts Subsetsum + Backtracking approach: While reconstructing the optimal solution from the 2D dp did not explore all possible paths.

+ 13 pts Correct solution, but run time is not O(ntk)

✓ **- 1 pts** **Minor mistake in code**

There is no check to see if col-subset[row-1]>=0 before accessing table[row-1][col-subset[row-1]] leading to index out of bound exception,

Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Final Exam - 100 pts total **Fall 2019, CU-Boulder**

Notes:

- *Due date: 6pm on Sunday, December 8, 2019*
- *Submit a pdf file of your written answers to Gradescope and one py file with your Python codes to Canvas. All Python solutions should be clearly commented. Your codes need to run to get credit for your answers.*
- *You can ask clarification questions about the exam in office hours and on Piazza. However, please do not ask questions about how to answer a specific question. If there is confusion about any questions, we will address those issues at the beginning of class on December 5.*
- *All work on this exam needs to be independent. You may consult the textbook, the lecture notes and homework sets, but you should not use any other resources. If we suspect that you collaborated with anyone in the class or on the Internet, we will enforce the honor code policy strictly. If there is a reasonable doubt about an honor code violation, you will fail this course.*

---

1. (10 pts) Consider the following merge() algorithm to merge two sorted arrays into a larger sorted array. There are three errors in the algorithm.

```
MergeWithErrors(A, p, q, r)
    low = A[p..q]
    high = A[q..r]
    i = 0
    j = 0
    k = p
    while(i < q-p+1 and j < r-q)
        if(low[i] <= high[j])
            A[k] = low[i]
            j++
        else
            A[k] = high[j]
```

Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Final Exam - 100 pts total **Fall 2019, CU-Boulder**

```
            i++
        k++
    while(i < q-p+1)
        A[k] = low[i]
        i++
        k++
    while(j < r-q)
        A[k] = high[j]
        j++
        k++
```

(a) (5 pts) List the three errors in the MergeWithErrors algorithm.
Solution.

1st error, high array should be indexed array[q+1 .. r]

2nd error, the first while loop conditional goes one index to short for the length of high sub-array. Should be $j < r - q + 1$

3rd error, in the first while loop, increment $i++$ or $j++$ is done wrong, they should be flipped.

Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

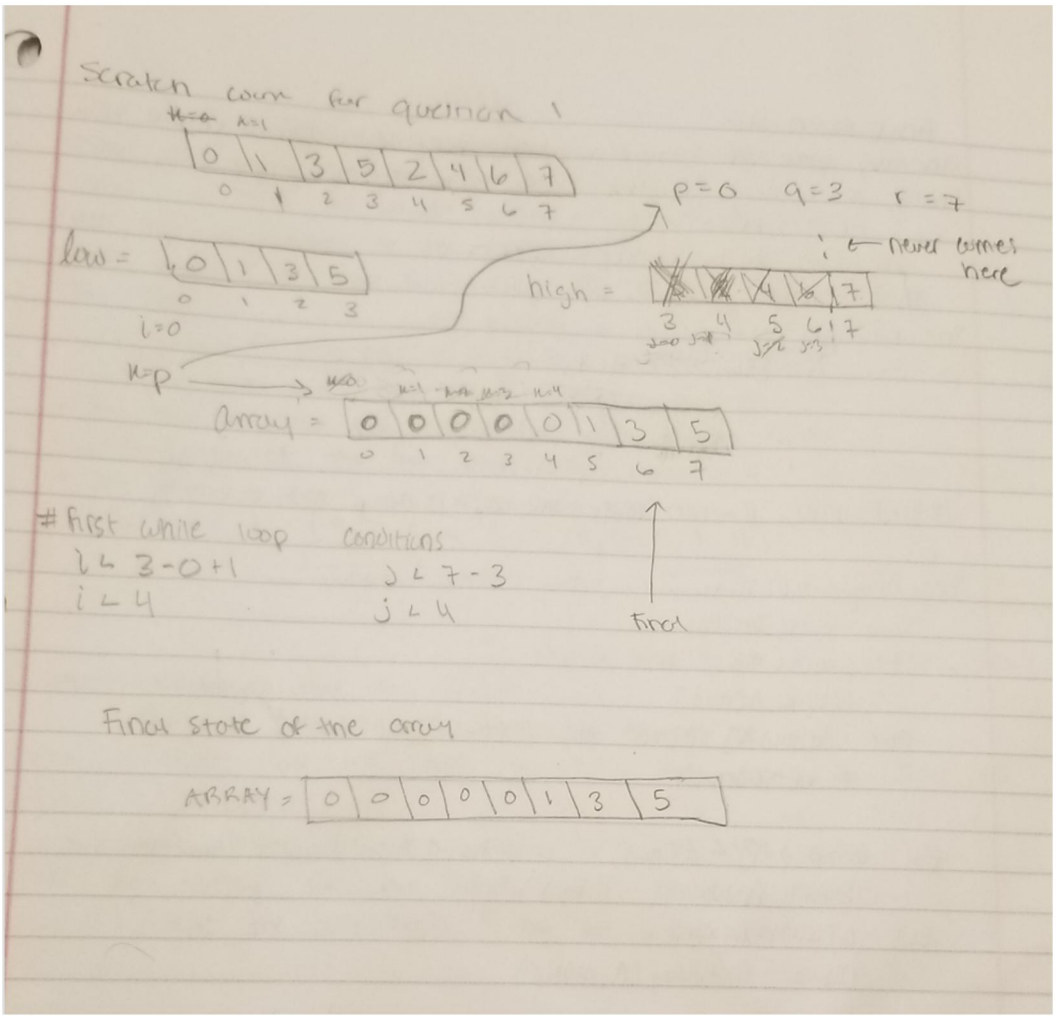
Final Exam - 100 pts total **Fall 2019, CU-Boulder**

(b) (5 pts) For the following call to MergeWithErrors, what is the state of the array A after running MergeWithErrors. You can assume that the size of A won't change and values written outside the indices of A will be lost.

$$A = [0, 1, 3, 5, 2, 4, 6, 7]$$

$$MergeWithErrors(A, 0, 3, 7)$$

Solution.



Name: Jonathan Phouminh

ID: 106054641

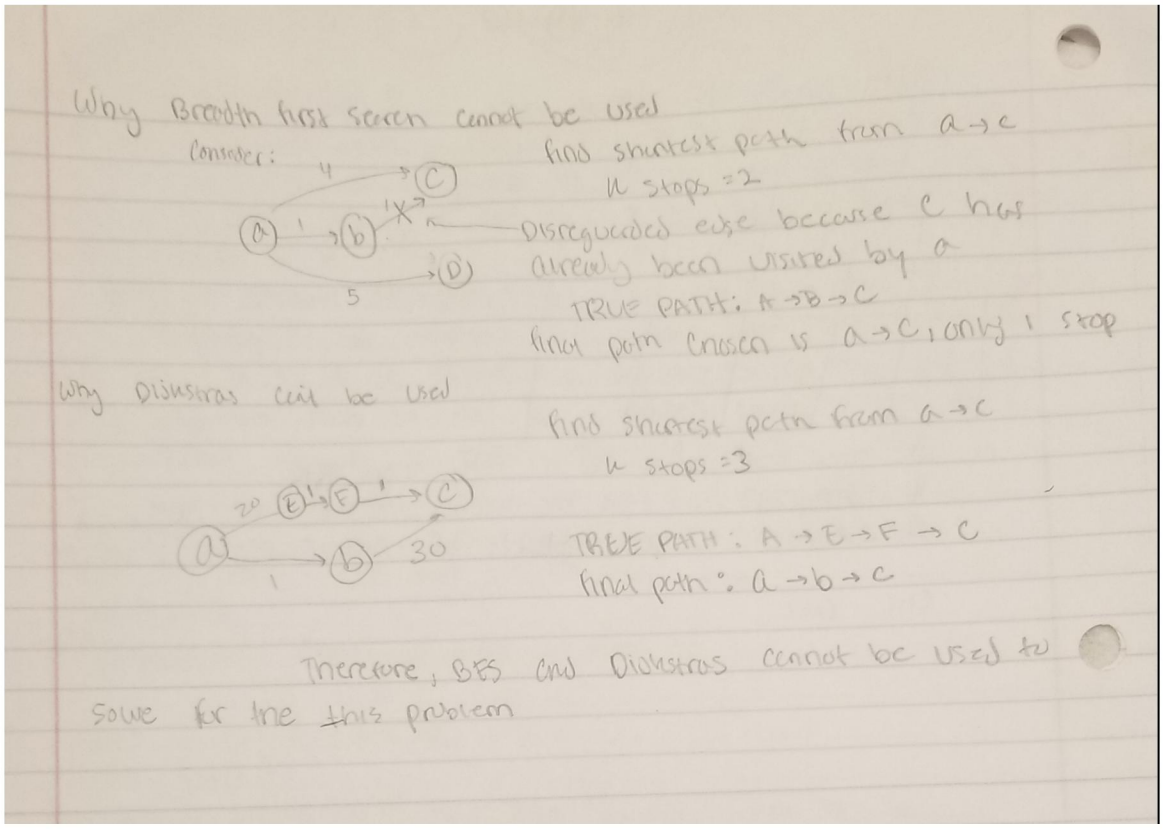
CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Final Exam - 100 pts total **Fall 2019, CU-Boulder**

2. (25 pts) Let $G = (V, E)$ be a directed weighted graph of the pathways on the CU-Boulder campus, with edge weights being distances between different buildings/intersections. Engineering and Humanities are two vertices of G, and $k > 0$ is a given integer. Assume that you will stop at every building/intersection you pass by. A shortest k-stop path is a shortest path between two vertices with exactly k stops.

   (a) (5 pts) Provide an example showing that the shortest k-stop path can't necessarily be found using Breadth-first search or Dijkstras algorithm. You need an example for each algorithm that shows where it fails.

Solution.



(b) (10 pts) Design an algorithm to find the shortest path from Engineering to Humanities that contains exactly k stops (excluding Engineering and Humanities). Notice that a k-stop path from these two buildings may not exist. So, your algorithm should also take care of such possibility. You need to provide an explanation of how your algorithm works to receive credit for this question.

Solution.

An Algorithm that could work for this is by executing a bellman ford algorithm that will have the source node be Humanities and Engineering be the Target.
We modify the loop of the Bellman Ford algorithm by forcing it to stop when k number of edges from source to all other reachable vertices regardless if Engineering is included in the set of reachable vertices.
Once we determine all the reachable vertices from source with k edges we will count determine if a path from engineering to humanities exists. If not, return False
Otherwise return the sum of all the edge weights of the path from engineering to Humanities

(c) (10 pts) Implement your algorithm using the starter code provided on Canvas.

3. (25 pts) To entertain her kids during a recent snowstorm, Dr. Hoenigman invented a card game called EPIC!. In the two-player game, an even number of cards are laid out in a row, face up so that players can see the cards' values. On each card is written a positive integer, and players take turns removing a card from either end of the row and placing the card in their pile. The objective of the game is to collect the fewest points possible. The player whose cards add up to the lowest number after all cards have been selected wins the game.
One strategy is to use a greedy approach and simply pick the card at the end that is the smallest. However, this is not always optimal, as the following example shows: (The first player would win if she would first pick the 5 instead of the 4.)

4 2 6 5

(a) (10 pts) Write a non-greedy, efficient and optimal algorithm for a strategy to play EPIC!. The runtime needs to be less than $\theta(n^2)$. Player 1 will use this strategy and Player 2 will use a greedy strategy of choosing the smallest card. **Note: Your choice of algorithmic strategy really matters here. Think about the types of algorithms we've learned this semester when making your choice.** You need to provide an explanation of how your algorithm works to receive credit for this question.

Solution.
we will attempt to find the optimal solution for player one by examining the best possible winning hand by looking one step ahead of what the opponent will take, then we will take the minimum of if the choice of picking the left card was less than the right card and vice versa.

```
counter 1 # these will calculate the possible scores if we
counter 2                              pick left or right

have trivial condition base case where we only have 2
elements, just pick the minimal.

First set of conditionals to calculate the best score if
we were to pick the card on the right
    examine the sumo f right end card and the minimal between
    the cards after taking into account of what greedy takes
```

```
Second set of conditions to do the same thing but considering
if we were to pick the card on the left

finally, take the card that resulted in the minimal score pissble.
```

(b) (15 pts) Implement your strategy and the greedy strategy in Python and include code to simulate a game. Your simulation work for up to 100 cards, and values ranging from 1 to 100. Your simulation should include a randomly generated collection of cards and show the sum of cards in each hand at the end of the game.

Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Final Exam - 100 pts total **Fall 2019, CU-Boulder**

4. (22 pts) In a previous homework assignment and classroom activity, we worked on the problem of finding the peak in an array, where array $A[1, 2, \cdots, n]$ with the property that the subarray $A[1..i]$ has the property that $A[j] > A[j+1]$ for $1 \leq j < i$, and the subarray $A[i..n]$ has the property that $A[j] < A[j+1]$ for $i \leq j < n$. For example, $A = [16, 15, 10, 9, 7, 3, 6, 8, 17, 23]$ is a peaked array.

   Now consider the *multi-peaked* generalization, in which the array contains k peaks, i.e., it contains k subarrays, each of which is itself a peaked array. Suppose that $k = 2$ and we can guarantee that neither peak is closer than $n/4$ positions to the middle of the array, and that the "joining point" of the two singly-peaked subarrays lays in the middle half of the array.

   (a) (8 pts) Now write an algorithm that returns the minimum element of A in sub-linear time.

   Solution.

```
The following algorithm is based on assumption that they work properly.
   findPeak(a,p,r) # modified function to make it return index of peak
   findValley(a,p,r) # function returns the value of the valley sub-array


def findMinValley(a,p,r): #given start and end index
    midpoint = findPeak(a,p,r) # returns index of peak
    left_valley = findValley(a,p,mid) # returns left vall value
    right_valley = findValley(a,mid,r) #returns the right_valley value

    return min(left_valley,right_valley)
```

(b) (10 pts) Prove that your algorithm is correct. (Hint: prove that your algorithm's correctness follows from the correctness of another correct algorithm we already know.)

Solution.
The one important detail of our algorithm that determines the correctness of our algorithm is ensuring that when find valley is called that it is given the correct-subarrays that contain only one valley in each.

Assume original array is not split at the peak original array and instead split at some other index.
If this holds then for some cases, the splicing could result in one sized sub-array that contains 0 valleys and the other containing 2 valleys or event the splitting could cause the riddance of one valley entirely (valley was picked as split).
This is a contradiction because our function is going to always return the minimum of 2 valleys in the original array.
Therefore our algorithm is correct by splitting the original array at the peak because it guarantees that the two sub-arrays that were spliced each contain one valley.
Once the two valleys are returned the algorithm returns the minimum of the two, therefore our algorithm is correct

(c) (4 pts) Give a recurrence relation for its running time, and solve for its asymptotic behavior.

Solution. This is runtime of findPeak and findValley

COST	TIME
c1	1
c2	1
c3	1
c4	1
c5	1
c6	1
c7	T(n/2)
c8	1
c9	T(n/2)

```
  T(1) = a
0 T(n) = T(n) + 1
1 T(n) = T(n/2) + 1 + 1
2 T(n) = T(n/4) + 1 + 1 + 1
.
.
k T(n) = T(n/2**k) + k(1)

n/2**k = 1 --> n = 2**k --> k = log(n)
```

T(n) in $\Theta(logn)$

Cost time of findMinValley

COST	TIME
c1	log(n)
c2	log(n)
c3	log(n)
c4	1

```
T(n) = c1*log(n) + c2*log(n) + c3*log(n)
T(n) = O(3logn) -->
```

T(n) $\in \Theta(log(n))$

5. (18 pts) Suppose we are given a set of non-negative distinct numbers S and a target t. We want to find if there exists a subset of S that sums up to **exactly** t and the cardinality of this subset is k.
   Write a python program that takes an input array S, target t, and cardinality k, and returns the subset with cardinality k that adds to t if it exists, and returns $False$ otherwise. Your algorithm needs to run in $O(nt)$ time where t is the target and n is the cardinality of S. In your code, provide a brief discussion of your runtime through comments, referring to specific elements of your code.

   For example -

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

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

   Input: s = {2,1,5,7}, t = 6, k = 3
   Output: False
   Explanation: No subset of size 3 sums to 6.