Final Study Guide

## Heaps

1. How do priority queues differ from regular queues? A Priority queue is a Binary Heap
2. What three methods are part of the priority queue ADT?

deleteMin or remove(), insert or add(), findMin or peek()

1. State the heap ordering property. Be as clear as possible when answering this question.

Parent must be smaller than its children if it’s a Min Heap

Parent must be larger than its children if it's a Max Heap

1. **What is a complete binary tree?**

Each level except the last is filled, and all nodes are as far left as possible.

1. A complete binary tree can be stored efficiently in an array. Given an index i in this array, how would you identify the index of a parent node, left child or right child node assuming:
   1. The root of the tree is located at index one

Parent node = index / 2

Left Child = index \* 2

Right Child = index \*2 + 1

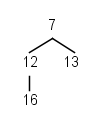
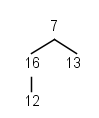
* 1. The root of the tree is located at index zero

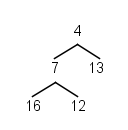
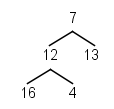
Parent: (index - 1) /2

Left: index \* 2 + 1

Right: index \* 2 + 2

1. Draw the resulting heap after adding the following elements from left-to-right: 16, 13, 7, 12, 4, 0, -2.  
   <http://prntscr.com/bchxvt>  
   <http://prntscr.com/bchz47>  
     
   

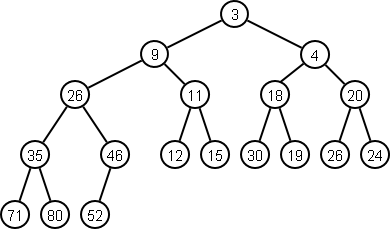




*Note: You should show the heap in tree-form after each add() operation above.*

*Note: Make sure to denote clearly each percolateUp() operation and how it was executed.*

1. For the following heap, show the steps involved when calling deleteMin().



*Note: Your solution should clearly show each step of the percolateDown() operation.*

Remove 3

Move 52 up swap 4, swap 18, swap 19

1. What is the difference between a min-heap and a max-heap?

A min heap has the parent as smaller than its children, and as a result the root will always be the smallest element of the heap.

A Max heap has parents that are larger than its children, and as a result the root will always be the largest element in the heap.

1. Name three algorithms we have discussed this quarter that use a heap as part of their solution.

Heap sort, selection algorithm (find kth largest), Dijkstra

## Sorting

1. What is an **inversion**

two elements out of order

1. What is the worst case running time of the following sorts.
   1. bubbleSort() O(n^2)
   2. selectionSort() O(n^2)
   3. insertionSort() O(n^2)
2. Write a method that performs the selection sort algorithm on an input array.
3. Write a similar method for insertion sort.
4. Which of the sorts from (2) have the best running time in the best case scenario? What is the big-oh in the best case? insertion sort O(n)
5. Given the sort chosen from (5), what type of input is necessary to get the best case scenario?

It is mostly sorted

1. Insertion sort as an algorithm has weaknesses depending on the type of input given to the algorithm.
   1. Explain how shell sort is an improvement over insertion sort.   
      Instead of comparing each element to its neighbor it examines elements separated by a distance.
   2. How does shell sort address the weakness of the insertion sort algorithm?  
      Insertion sort has a weakness that if the input is in reversed order, it has to iterate and shift each element, but with shell sort it can use the gaps to quickly put the input in a mostly sorted order before performing insertion sort. Significantly cutting down on the number of swaps that have to happen.
2. Merge sort and quick sort are called divide-and-conquer algorithms. Explain how divide and conquer algorithms work.

They break up the list into sub lists to focus on sorting a smaller portion before going out and sorting all of the sub lists together

1. Show the recursive calls and merge steps for the merge sort algorithm on the following input: [99, 22, 117, 41, 98, 110, 1, 22].

Len = 8

[99, 22, 117 41] , [98, 110, 1, 22]

[99,22],[117, 41], [98, 110], [1, 22]

[99], [22] [117] [41] [98], [110], [1], [22]

[22, 99], [117] [41] [98], [110], [1], [22]

[22, 99], [41, 117] [98], [110], [1], [22]

[22, 99], [41, 117], [98, 110] , [1], [22]

[22, 99], [41, 117] , [98, 110], [1, 22]

[22, 41, 99, 117], [98, 110] [1, 22]

[22 41 99 117], [1, 22, 98, 110]

[1, 22, 22, 41, 98, 99, 110, 117]

1. For the merge sort algorithm:
   1. Is it possible to get stack overflow for any input array? It is hard to get an overflow error unless you have insanely huge input, and even then it would likely only have 10 stacks
   2. Does the algorithm use extra space to perform the sort? Yes N extra space
   3. What is the worst case AND average case big-oh? Worst - o(n logn) Average O(n logn)
2. Show the steps of the quick sort algorithm on the following input: [6, 5, 12, -4, 33, 18, 2, 3, -6, 10, 3, 1]

[5 12 -4 33 18 2 3 -6 10 3 1]

I =5 ->12

J = 1

Swap

[5, 1, -4, 33, 18 2, 3 -6 10 3 12]

I = -4 →33

J = 3

Swap

[5, 1 ,-4 , 3 18 2, 3, -6, 10 33 12]

I = 18

J = 10 → -6

Swap

[5, 1 -4 3 -6, 2 3, 18, 10, 33, 12]

I = 2 -->3 → 18

J = 3

Crossed so split

[5, 1, -4, 3, -6, 2, 3], 6, [18, 10, 33, 12]

*Note: Use the first element as your pivot for each step of the algorithm.*

*Note: Make sure to clearly denote your parititioning step, with indices i and j (as shown in class).*

*Note: You can declare that your work uses a well known sort (such as insertionSort()) on any sub-arrays of size three or smaller.*

1. Explain how median-of-three partitioning works for pivot selection.

You pick an element at the end, beginning and in the middle compare the 3 and use the value in the middle for your pivot.

1. Give an example input array where median-of-three does not adequately partition the array.

[ 1, 4, 3, 7, 2, 6 , 0, 10]

1. For the quick sort algorithm:
   1. Is it possible to get stack overflow for any input array? Yes if you selected pivots poorly
   2. Does the algorithm use extra space to perform the sort? No extra space needed
   3. What is the worst case AND average case big-oh? Worst - O(n^2) Average - O(n logn)
2. What does it mean that heap sort can be programmed as an **in-place sort**?

In-place means that you move elements around in the existing array without having to store elements in a temp data structure.

Storing the element, you removed with DeleteMax, in the space at the end where you had the last element of the heap before you percolated down. This will create a sorted array after you’d performed DeleteMax on all the elements in the Heap.

1. Explain how the in-place heap sort routine works.
2. What is the minimum amount of work necessary for any comparison based sort. ie. given a comparison sort f(n), give a g(n) such that f(n) = Ω(g(n)).  
   n log(n)

## Graphs

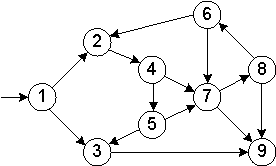
1. Match each of the following terms with their corresponding definitions:

|  |  |
| --- | --- |
| Term | Definition |
| Weighted graph | A graph that includes a value on each edge. These values can represent distance, cost or some other numerical quantity. |
| Connected graph | A graph where for every pair of vertices u and v, there is a path from u to v. |
| Path | A linear sequence of edges in a graph that allow you to traverse from a source node to a destination node. |
| Cycle | A path that begins and ends at the same vertex. |
| Incident | Given a vertex u, all edges (u, v) from u to another vertex v are said to be \_\_\_\_\_\_\_\_\_\_\_\_ to u. |
| In-degree | The number of incoming edges to a vertex. |
| Vertex | The elements we store in a graph structure, along with their relationships. |
| Edge | Represents a relationship between two vertices in a graph. |
| Sub-graph | Given a graph G = (V, E), this type of graph contains a subset of V and E. |
| Adjacent | Two vertices u and v are said to be \_\_\_\_\_\_\_\_\_ if there exists an edge (u, v). |
| Out-degree | The number of outgoing edges from a vertex. |
| Directed graph | A graph where the edge (u, v) is not the same as (v, u). |
| Complete graph | A graph that contains an edge between every pair of vertices. |

1. What is a **DAG**? Directed Acyclic (No cycles) Graph (ie Tree)
2. Explain how adjacency lists are used to store vertices and edges in a graph structure. Be specific.

You create an array with a space for all of the nodes (|v|) in the graph, and include a list of all of the nodes connected to that node through and edge. Often stored as a linked list similar to a chaining in a hash table.

1. Which of the following graph representations uses more memory in the worst case? Justify your answer.
   1. Adjacency list - A complete graph would be difficult to search because each vertices it connected to all other vertices and would have a linked list of n-1 stored in each index
   2. Adjacency matrix - This will take a constant amount of space whether completely empty or full, and once the graph becomes large and approaches a complete connected graph this can take up less space since its indicating with a 0 or 1 whether an connection exists.
2. Which of the following graph representations uses more memory in the best case? Justify your answer.
   1. Adjacency list - When there are few edges between Vertices then they can be better represented with an Adjacency list. If you want to find adjacent vertices you can look up one vertex and go through the linked list to see if the node you are looking for is connected
   2. Adjacency matrix - This will take up space storing two spots for each adjacency possibility. This means that a graph with 4 vertices would have 16 possible spots even if there are only 6 edges. When we get into larger graphs this can mean a lost of wasted space if the graph does not have many edges.
3. Given the following graph representations, what is the runtime cost of searching for an edge (u, v) in a graph?
   1. Adjacency list O(|e|/|v|)
   2. Adjacency matrix O(1)
4. Show the depth-first search for the following graph:



*Note: Make sure to show each step of the algorithm, including any backtracking that occurs.*

Start at 1

Goes to 2

Goes to 4

Goes to 5

Goes to 3

Goes to 9, no out going from 9

Reverse to 3 → 5

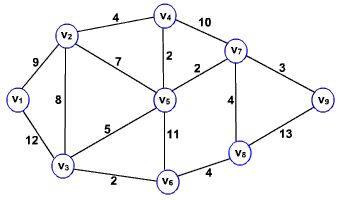
Goes to 7

Goes to 8

Goes to 6

Reverse 8, 7, 5, 4, 2, 1

1. For the following graph, show the shortest path from V9 to all other nodes using Dijkstra's algorithm: O(|V|\*log |v|)



*Note: Your solution should show each update performed during the algorithm.*

*Note: Your answer should include the shortest path cost for each vertex Vi as well as the path leading from V9 to Vi.*

*Visit v9*

*V7 = 3, v8 = 13*

*Visit v7*

*V8 = 7 , v4 = 13, v5 = 5*

*Visit v5*

*V4 = 7, v2 = 12, v3 = 10, v6 = 16*

*Visit v8*

*V6 = 11*

*Visit v4*

*V2 =11*

*Visit v3*

*V1 = 22*

*Visit V6*

*Nothing updates*

*Visit V2*

*V1 = 20*

*Visit V1*

*Shortest path calculated*

*V1 = 20*

*v2 = 11*

*v3 = 10*

*v4 = 7*

*v5 = 5*

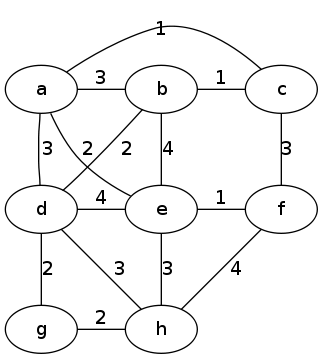
*v6 = 11*

*v7 = 3*

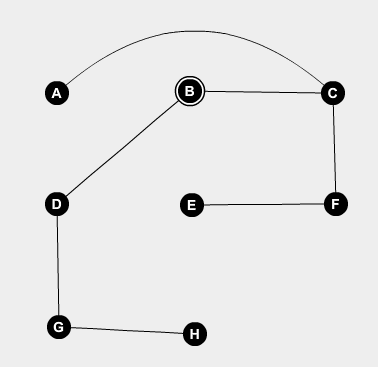
*v8 = 7*

*v9 = Home*

1. What type of structure would we typically use to store vertices and their current distance values as part of Dijkstra's algorithm? Heap
2. When updating vertex distances during Dijkstra's algorithm, how would you update the position of a vertex in the structure from (7). Decrease key or increase key O(logN)
3. Is each shortest path generated by Dijkstra's algorithm unique? NO. ie. given a shortest path from u to v, can another path be found in the same graph with the same total cost as the shortest path found by Dijkstra's? yes you can find paths with the same total cost and if you do it doesn't matter which one you select.
4. Show a minimum spanning tree for the graph below using Prim's Algorithm:

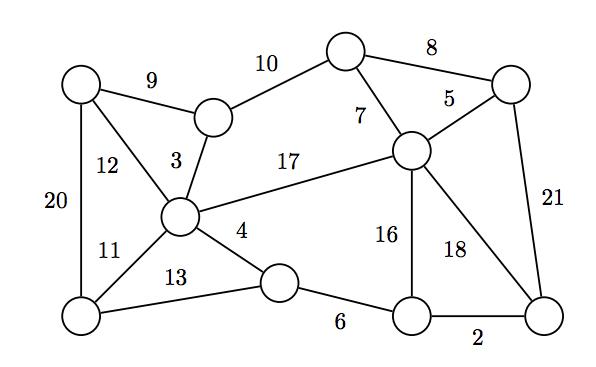


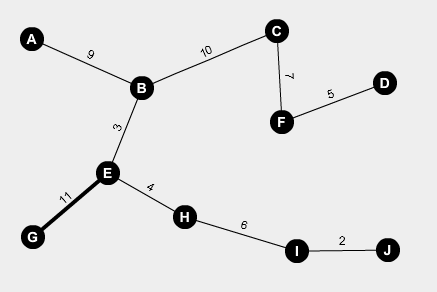
Start with f  
Add e-f (1)  
Add c-f (3)  
Add b-c (1)  
Add a-c (1)  
Add b-d (2)  
Add d-g (2)  
Add g-h (2)



*Note: Your solution should show the set of known and unknown vertices for each step of Prim's algorithm.*

1. Show a minimum spanning tree for the graph below using Kruskal's Algorithm:



Add Edge 2  
Add Edge 3  
Add Edge 4  
Add Edge 5  
Add Edge 6  
Add Edge 7  
Add Edge 9  
Add Edge 10  
Add Edge 11  


*Note: You do not need to show the steps of the union-find algorithm. Instead show how each edge is chosen as part of the algorithm.*

1. Is the minimum spanning tree in a graph unique always unique? Explain when this will or will not be the case.

## Algorithm Design Techniques

1. For each of the following categories of algorithms, give a description of how that technique is used to solve a problem:

|  |  |
| --- | --- |
| Design Technique | Description |
| Brute Force Algorithm | Straight-forward, simple, exhaustive. Goes through all elements. If you asked a kid to solve the problem this is the method they would use. |
| Greedy Algorithm | Takes the next best choice. Short Sighted |
| Decrease and Conquer Algorithm | Reduce problem size by constant, 1, or a variable |
| Divide and Conquer Algorithm | Break up into smaller problems by a constant |
| Transform and Conquer Algorithm | Change the representation of an input and solve it |
| Dynamic Programming | Solve a specific subproblem |
| Backtracking Algorithm | Recursively solve problem, find candidate, if stuck back track. Method calls |

1. For each of the following categories, give two example algorithms we have studied this quarter that use the technique:

|  |  |
| --- | --- |
| Design Technique | Algorithms |
| Brute Force Algorithm | Selection Sort, Sequential Search |
| Greedy Algorithm | Kruskal, Prim’s |
| Decrease and Conquer Algorithm | Topological, Insertion Sort, Binary Search |
| Divide and Conquer Algorithm | Quicksort, Mergesort |
| Transform and Conquer Algorithm | Heapsort, Presorting, Breadth First Search |
| Dynamic Programming | Fibonnachi, Coin Changing |
| Backtracking Algorithm | Tree Traversals (pre-order, in-order, post-order), DFS |

1. State the definition of the travelling salesman problem.   
   Given a set of cities and the distance between the cities, vertices represent cities and edges are the distances between cities. Give the shortest route for the salesman to visit all cities exactly once and return to the city in which they originated. This is a hamiltonian circuit. O(n!) because when leaving the first city you have n-1 options and the second city has n-2 and so forth
2. Give a big-oh estimate of the brute force solution to the travelling salesman problem. Justify your answer.
3. Give a description of a greedy algorithm that, given coin denominations 1, 5, 10, 25, will determine the minimum number of coins needed to make N cents.
4. Show the steps of the topological sort algorithm on the following graph:

1. Compute In degree of each node.

2. Create a Queue

3. Push all vertices whose in degree is zero into Queue.

run a loop until number of nodes.

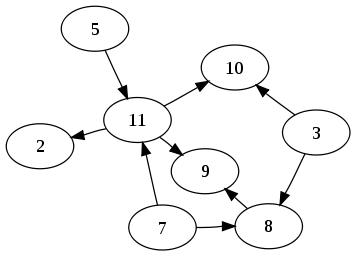
4. Pop a vertex from Queue. print this vertex.

5. For each child of vertex decrement its in\_degree by one.

6. if any node having in\_degree = 0 then push it onto Queue because now the child is ready to execute.

7. Repeat the process until TRUE.

The above approach will not lead any starvation to any vertex.



1. Explain why it is necessary for a graph to be a DAG before performing topological sort.
2. Explain how you would use a priority queue (heap) as part of the topological sort algorithm. How are the keys sorted in the heap?
3. Name two types of problems that, when represented with a graph, could benefit from the results of topological sort.
4. Given an array of size N, give an algorithm that uses dynamic programming to determine the minimum number of coins it takes to make N cents with denominations C1,C2, C3, … ,Cm.

Note: You may not use any extra space beyond the array of size N.

Note: List the steps of your algorithm clearly, or give psuedocode as an answer.

1. State the definition of the eight-queens problem.

Explain how an algorithm can use recursive backtracking to solve the eight-queens problem.