Exam 2 Review

Review Problems Covered in Workshop + More

# Stacks

1. Linked List Implementations
   1. push()

A screen shot of a computer program

Description automatically generated

* 1. pop()

A computer screen with colorful text

Description automatically generated

* 1. top()

A screen shot of a computer code

Description automatically generated

1. Array-Based Implementations
   1. push()

A computer code with text

Description automatically generated

A black text on a white background

Description automatically generated

* 1. pop()

A computer code with text

Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. top()

A screen shot of a computer program

Description automatically generated

A computer code with text

Description automatically generated with medium confidence

1. Infix to Postfix (Implementation & Tracing)
2. Postfix to Infix (Implementation & Tracing)
3. Evaluate Postfix (Implementation & Tracing)
4. Given an expression, find out whether or not that expression contains valid parentheses. An expression contains valid parenthesis if it has the proper parenthesis ( ), [ ], { } in the correct order.

string = “{ [ ] ( ( ) ) }” → true string = “{ ( ] ) } }” → false

bool validParantheses(string s){

}

A screen shot of a computer program

Description automatically generated

1. Reverse a stack using recursion (you are only allowed access to the stack passed in through your function, no additional data structures allowed)

void reverseStack(stack<int>& s){

}

A screen shot of a computer program

Description automatically generated

1. Given a value, return the index of where it is in the stack (assume the top of the stack is index 0, and the bottom of the stack is index n - 1, n being the size of the stack). If the value doesn’t exist in the stack, return -1 (iteratively and recursively) int returnIndex(stack<int> &s){

}

A screenshot of a computer program

Description automatically generated

# Queues

1. Linked List Implementations
   1. enqueue()

isEmpty() =

front == nullptr

A screen shot of a computer program

Description automatically generated

* 1. dequeue()

A black text on a white background

Description automatically generated

A computer code on a black background

Description automatically generated

* 1. front()

A black background with white and pink text

Description automatically generated

1. Array-Based Implementations
   1. enqueue()

A computer screen with text and numbers

Description automatically generated

* 1. dequeue()

A screen shot of a computer code

Description automatically generated

* 1. front()

A black text on a white background

Description automatically generated

1. Priority Queues
   1. enqueue()

A screen shot of a computer program

Description automatically generated

1. What are the differences between a queue and priority queue?

A close up of a text

Description automatically generated

1. Implement a function that takes in two arrays, one that holds a list of tasks, and another that holds each task’s associated time to complete and performs round-robin scheduling on the list of tasks (quantum time is the amount of time to perform a task for before moving on to the next). Output the tasks in the order that they’re completed.

void roundRobin (string tasksArr[], int taskTimesArr[], int quantumTime, int size){

}

A computer screen shot of a program code

Description automatically generated

1. Implement a function that reverses a queue recursively (you’re not allowed other data structures, only have access to the queue STL functions - push(), pop(), front(), size()...) void reverseQueue(queue<int>& q){

}

A screen shot of a computer code

Description automatically generated

1. Implement a function that returns the number of even elements in a queue (you cannot use any additional data structures. Elements in the queue should as well be in the same order as when passed through the function after the function is done running)

int countEven(queue<int> &q) {

}

A screen shot of a computer code

Description automatically generated

# Sorting

1. Merge, Quick, Shell, Bucket and Heap Sort (Implementations, Tracing and Time Complexities)

A table with text on it

Description automatically generated

1. Sort the following array, [7, 4, 9, 13, 1, 5], using Merge, Quick, and Shell Sort

Merge Sort: A piece of paper with writing on it

Description automatically generated

Shell Sort:

A close-up of a piece of paper

Description automatically generated

1. Sort the following array, [541, 847, 192, 316, 729], using Bucket Sort

A paper with writing on it

Description automatically generated

1. Build a Min Heap from the following array, [6, 3, 5, 9, 8, 4]. After doing so, perform Heap Sort to it.

A notebook with writing on it

Description automatically generated

# Hashing

1. Direct Hashing, Linear/Quadratic Probing, Double Hashing and Separate Chaining

(Implementations and Tracing)

A black screen with text on it

Description automatically generated

A computer screen shot of a code

Description automatically generated

A computer screen shot of a code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

1. Insert the following values into a Hash Table of size 5 using Double Hashing

10, 20, 15, 30

Index = hash1() + ( i \* hash2() ) % tableSize

hash1() → value % tableSize hash2() → 3 - (value % 3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 10 | 20 |  | 15 | 30 |

# A notebook with math equations Description automatically generated

# Heaps

1. Building Max/Min Heaps
2. Build a Min Heap from the following array, [10, 2, 5, 3, 7, 4]

A diagram of a number

Description automatically generated

1. Build a Max Heap from the following array, [2, 4, 7, 3, 8, 5]

A diagram of numbers and arrows

Description automatically generated