Test 2 Study

(1) Challenge

Topics:

Data Structures

- Heaps
- Stacks
- Queues
- Hashing (https://canvas.uh.edu/courses/1213/files?preview=1076002)
    - Direct Hashing
    - Linear Hashing
    - Quadratic Probing
    - Double hashing
    - Chaining
        - HashFunction
- B-Trees (simple problems?) [NOT on EXAM2]
    - Traversals
        - Pre Order (LRN)
        - In Order (LNR)
        - Post Order (NLR)
    - Depth (to root)
    - Height (to leaf)
    - Patterns
        - DFS
        - BFS

Sorting Methods + Big O's

- HeapSort
- ShellSort
- MergeSort
- QuickSort
- Bucket Sort

Coding to know

- MergeSort (recursively)

- ShellSort
- Heapsort
  - heapify?
- All basic structures + basic functions
  - Insertion
  - Delete
  - Search
  - isEmpty()

Study

problems

1. infix ⟶ postfix
2. postfix ⟶ infix
3. evaluate postfix

easy points

1. postfix ⟶ infix (both ways)
2. everything in hashing (know them in code)
3. everything in sorting
4. everything in heaps
5. valid parenthesis

Stacks

Stacks using an array

- pointer top
  Ex. [3, 2, 5, 2, 5, 4]
  ^top
  pop() ⟶ decrement top
  push() ⟶ increment top
  - only if top > MAX_SIZE Array

Using recursion as a stack (ON THE EXAM p SURE) know for a queue/linkedList/ stack

Reverse a queue, WITHOUT using a stack

- Steps:
  - Figure out how to do with stack ⟶ do with recursion
    recurse(q)

```
if (x == 0) {
return;
}
else {
int temp = q.front()
q.dequeue()
recurse(q)
q.enqueue(temp)
}
```

reverse a stack using recursion, no additional stacks

- I took a picture, write it in code and see for yourself

Stacks using a LL

push() → insert head
pop() → delete from the head

```
problem: Implement three stacks using one array
```

- use three top pointers (top 1, top2, top3)
  - split the array into 3 sections (size/3)

  ### write push1()

  - top1++ (check it does not pass size/3)

Applications of Stacks

- Expression matching **is different than** computation
  - Postfix → Infix (pretty easy just practice some hard examples)
  - Infix → Postfix (usually given 2 functions **isOperand()** and **priority()**)
    - (0) Make the stack and *str postfix*
    - (1) Read Expression L→R (use a forLoop)
    - (2) if *operand* add to result expression
    - (3) if *operator*
      - if stack empty → push
      - else → check **priority**
        - if priority of top is **>=** then we pop till empty or lower
        - else → push
    - (4) empty the stack at the end if anything is left
    - for these problems use three columns

- (1) character (2) stack (3) expression

- Reversals
  - Reversing a stack, linkedList, array, etc.
    - just add to the stack, and then pop()

```
void postToIn(str input){
stack s;
for (i=0; i < input.size(); i++){
if isOperand(input[i]){
s.push(input[i])
}
else {
op2 = s.top() order is opposite REMEMBER
s.pop()
op1 = s.top()
s.pop()
s.push(op1 + input[i] + op2)
}
}
}
```

if for the above you had to evaluate

- write a evaluate function
  - op1 (some switch function to get operator) op2

Hashing

```
for (i=0; i < size, i++){
index = (x+i) % size only thing that changes for linear/quadratic/double
if table[index] == -1:
table[index] = x;
return;
}
```

Queues

Queue :

```
[1 2 4 5 6] Problem: add (3) and keep sorted order
```

- think like dequeue() and enqueue the same element until correct placement arrives

    - for loop that runs the length of the queue

round robin

- while (!queue.isEmpty())
    - temp = q.front();
    - q.deque();
        - if (temp-cost) > 0
            - q.enqueue(temp-cost);
        - else { cout << "finished task# " << name of task << endl;}
- cout << "done with round robin";

Final Words

1. Stacks
    1. postfix
    2. infix
    3. valid parenthesis
    4. recursion in place of stack
2. Queues
    1. Round Robin
    2. Circular algorithms
        1. for this just think, i want queue in order
3. Hashing (code and by hand)
    1. Linear
    2. Quadratic
    3. Double
    4. Chaining
4. Heaps **do not need to know code for heapify function**
    1. Traverse, tracing
    2. heapify() is called on last internal node
5. Sorting
    1. Merge, Quick, Shell, Bucket, HeapSort

Heaps

Know the tracing, tree method tracing, and building with general blocks
how to build a heap??

```
for ( i = (size/2)-1 ; i > 0; i++){
heapify(i);
}
```

heap sort?

(do the below n times) keep an end pointer (end--)

```
swap(a[0],a[end]);
heapify(a[0]);
```

how does heapify work?

max heap: largest element selected in a triangle, swap with root → call on swapped element

Using arrays

stacks using an array

top pointer, add left to right

queues using an array

front and back pointer (increment by modulo)

keep track of size

if size == size of array → cannot add

infix to postif w/parenth

look it up on slides

Hashing

```
for (O → N){
index = SOMETHING
if (arr[index] == -1){
arr[index] = x;
return;
}
}
```

Direct

Something = i % size

Linear

Something = (hash(x) + i) % size

Quadratic

Something = (hash(x) + i^2) % size

Double

Something = (hash1(x) + i *hash2(x)) % size*
know how to trace by hand*

Chaining

nodes, insert at start, look at code online, on slides

Sorting

Shell

1. select gaps
    1. (1) size
    2. and so on i forgot