Exam 2 practice questions

1.Write a function that takes in two stacks of varied length and returns a linked list contains sum of the other stacks numbers. if one stack is longer than the other then the remaining digits should just be added to the linked list at the end.

Example:

- stack A:(top) 9 → 6 → 4 → 4 → 2 → 1
- stack B:(top) 10 → 9 → 8 → 3 → 0 → 6 → 7
  return
- linkedlist C: (head) 19 → 15 → 12 → 7 → 2 → 7 → 7

```
Node* addTwoStacks(stack st1, stack st2){}
```

```
Node* addTwoStacks(stack st1, stack st2) {
    linkedlist temp;
// assume peek() return the top node, pop() return the value that being
deleted
    while(st1.peek() != nullptr || st2.peek() != nullptr) {
        temp.addAtEnd(st1.pop() + st2.pop());
    }
    while(st1.peek() != nullptr) {
        temp.addAtEnd(st1.pop());
    }
    while(st2.peek() != nullptr) {
        temp.addAtEnd(st2.pop());
    }
    return temp.getHead();
}
```

2.Write a function to print the diagonal in a dynamic array of queues.

Example: 0: [1, 2, 3, 5]
1: [2, 6, 9, 1]
2: [3, 7, 2, 9]
3: [6, 2, 7, 1]

Output: 1, 6, 2, 1

```
void printDiagonal(queue arr[], int size){}
```

```
void printDiagonal(queue arr[], int size) {
    for (int i=0; i<size; i++) {
        for (int j=0; j<i; j++)
            arr[i].dequeue();
        cout << arr[i].front() << endl;
    }
}
```

3. Implement a function to traverse a 2D array in a row order spiral form using stack and queue.

Example: [6, 2, 8, 1]
[3, 7, 5, 0]
[9, 3, 6, 2]
[5, 0, 3, 4]

Output: 6, 2, 8, 1, 0, 5, 7, 3, 9, 3, 6, 2, 4, 3, 0, 5

```
void spiralTraversal(int matrix[][4], int row, int col){}
```

```
void spiralTraversal(int matrix[][4], int row, int col) {
    bool flag = true;
    stack st;
    queue q;
    for (int i=0; i<row; i++) {
        if (flag) {
            for (int j=0; j<col; j++)
                q.enqueue(matrix[i][j]);
            while (!q.isEmpty())
                cout << q.dequeue() << endl;
        }
        else {
            for (int j=0; j<col; j++)
                st.push(matrix[i][j]);
            while (!st.isEmpty())
                cout << st.pop() << endl;
        }
        flag = !flag;
    }
}
```

4. Implement a recursive linked list add at a given position. Take in a position (index) as input and add x at that position (index).

```
void addPos(node* cu, int x, int pos){}
```

```
void addPos(node* cu, int x, int pos) { // pass in head as node* cu
    if (pos > size)
        return;
    else if (pos == 0) {
        node* temp = new node();
        temp->data = x;
        temp->next = head;
        head = temp;
        size++;
    }
    else if (pos == 1) {
        node* temp = new node();
        temp->data = x;
        temp->next = cu->next;
        cu->next = temp;
        size++;
    }
    else {
        pos = pos -1;
        addPos(cu->next, x, pos);
    }
}
```

5. Write a function that return the infix expression for a given postfix expression.
Given operand will only contains single digit/character.

```
string postfixToInfix(string exp) {
    stack st;
    for (int i=0; i<exp.length(); i++) {
        if (isOperand(exp[i]))
           st.push(string(1, exp[i]));
        else {
            string var1 = st.pop();
            string var2 = st.pop();
            st.push("(" + var2 + exp[i] + var1 + ")");
        }
    }
    return st.pop();
}
```

6. Implement a sorting function that have the time complexity as follow

   Best: O(nlog(n))

   Average: O(nlog(n))

   Worse: O(n^2)

- Quick sort

```
int partition(int arr[], int low, int high) {
    int mid = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < mid) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int mid = partition(arr, low, high);
        quickSort(arr, low, mid - 1);
        quickSort(arr, mid + 1, high);
    }
}
```

7. Describe the worse case scenario for quick sort.

Worse case O(n^2)
When the pivot is at the beginning, or the end of the array and the array is already sorted

8. Implement a hashing function that hash a large amount of data into the hash table. Since we don’t know how many element will be hashed, the hash table should never be full and can always store new hashed element. Implement this hashing function.

- Separate Chaining

```
struct node {
  int data;
  node* next;
  node (int x) {
    data = x;
    next = nullptr;
  }
};

void hashing(node* table[], int x) {
  int index = hash1(x);
  node* temp = new node(x);
  if (table[index] == nullptr)
    table[index] = temp;
  else {
    node* cu = table[index];
    while (cu->next != nullptr)
      cu = cu->next;
    cu->next = temp;
  }
}
```

9. Given the array {11, 9, 2, 23, 17, 14}, hash the array into a hash table of size 12 with hash1 = x % 12. If a collision happen, use hash2 = 11 – (x % 11) to look for an empty index. Write the hash table manually.

0:

1:

2: 2

3:

4:

5: 17

6:

7: 23

8:

9: 9

10: 14

11: 11

10. Implement a search function for hash table with hash1 = x % 15 using linear probing.
Time complexity for this function should be as follow:

- Best case O(1)
- Worse case O(n)

Return true if x is in the table, return false otherwise.

```
bool search(int table[], int x, int size){}
```

```
bool search(int table[], int x, int size) {
    int index = hash1(x);
    if (table[index] == x)
        return true;
    else {
        int hash1 = index;
        for (int i = 1; i < size; i++) {
            index = (hash1 + i) % size;
            if (table[index] == x)
                return true;
        }
    }
    return false;
}
```

11. Implement a function to find the n-th smallest element in an array using priority queue. You must implement both this function and enqueue function for priority queue.

```
void enqueue(int value){}

int nthSmallest(int arr[], int size, int n){}
```

- Enqueue

```
void enqueue(int value) {
    node* temp = new node();
    temp->data = value;
    temp->next = NULL;

    if (isEmpty()) {
        front=temp;
        rear=temp;
    }
    else {
        node* cu = front;
        node* prev = NULL;
        if (temp->data<cu->data) {
            temp->next = front;
            front = temp;
        }
        else {
            while (cu!=NULL && temp->data>=cu->data) {
                prev = cu;
                cu = cu->next;
            }
            prev->next = temp;
            temp->next = cu;
            if (temp->next==NULL)
                rear = temp;
        }
    }
}
```

```
int nthSmallest(int arr[], int size, int n) {
  priority_queue q;
  for (int i=0; i<size; i++) {
    q.enqueue(arr[i]);
  }
  for (int i=1; i<n; i++) {
    q.dequeue();
  }
  return q.front();
}
```

12. Build the min/max heap for the following array {8, 15, 9, 3, 12, 5}. Element in the array will be added to the heap one by one starting from index 0.

Min: {3, 8, 5, 15, 12, 9}

Max: {15, 12, 9, 3, 8, 5}

13. Give the min/max heap after executing these functions.

Add(7)

Add(19)

Add(3)

Add(15)

Add(8)

Pop()

Pop()

Add(9)

Add(5)

Min: {5, 8, 15, 19, 9}

Max: {9, 8, 3, 7, 5}

14. Implement a function that check if an array is a min heap. Return true if the array represent a min heap, return false otherwise.

```
bool isMinHeap(int arr[], int size) {
    for (int i = 0; i < (size - 2) / 2; i++)
        if (arr[i] > arr[2 * i + 1] || arr[i] > arr[2 * i +
2])
            return false;
    return true;
}
```

15. Implement a helper function that complete the sorting method below.

```
void sort(int array[], int begin, int end)
{
    if (begin >= end)
        return;

    int mid = begin + (end - begin) / 2;
    sort(array, begin, mid);
    sort(array, mid + 1, end);
    helper(array, begin, mid, end);
}

void helper(int array[], int begin, int mid, int end){}
```

```
void helper(int arr[], int begin, int mid, int end) {
    int i, j, k;
    int n1 = mid - begin + 1;
    int n2 = end - mid;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[begin + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    i = 0; j = 0;k = begin;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

16. Implement a void function that deletes a node in a singly linked list, given the value in the parameter.

```
struct node {
        int data;
        node* next;
};

void delete_node(node* head, int val) {

}
```

```
void delete_node(node* head, int val) {
  node* current = new node();
  current = front;
  node* prev = new node();
  prev = nullptr;
  if (current -> data == val) {
    front = current -> next;
    delete current;
    return;
  }
  else {
    while (current -> data != val) {
      prev = current;
      current = current -> next;
    }
    if (current == nullptr) {
      return;
    }
    prev -> next = current -> next;
    delete current;
  }
}
```

17. What is the output after running the following program?

```
int main()
{
    deque <int> dq;
    dq.push_back(10);
    dq.push_front(20);
    dq.push_back(30);
    dq.push_front(15);
    dq.push_back(60);
    dq.push_front(90);



    dq.pop_front();
    dq.pop_back();
    dq.pop_back();



    for (int i = 0; i < dq.size(); i++) {
        cout << dq.front() << " ";
        dq.pop_front();
    }

    return 0;
}
```

15 20 10

18. Implement a void Shell Sort function that has parameters of the array and the number of elements in the array.

```
void shellSort(int arr[], int n) {

}
```

```
void shellSort(int arr[], int n)
{
    for (int gap = n/2; gap > 0; gap /= 2)
    {
        for (int i = gap; i < n; i += 1)
        {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];
            arr[j] = temp;
        }
    }
}
```