CS 330 Intro to the Design and Analysis of Algorithms

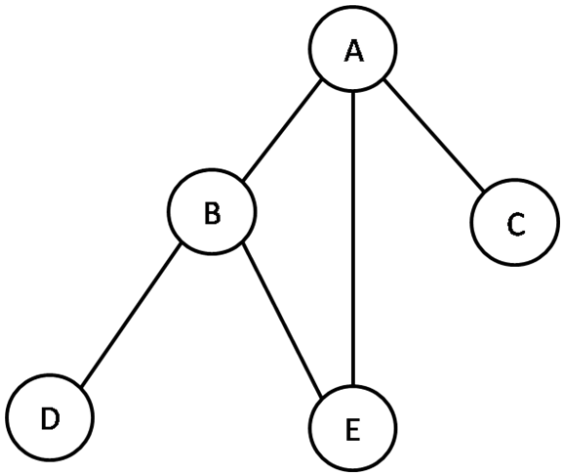
Homework 3

(20 pts)

1. Build a max-heap from the following input sequence: (22, 15, 36, 44, 10, 3, 9, 13, 29, 25, 2, 11, 7, 1, 17). You need to show all the intermediate steps. [6 points]

2. Illustrate the performance of heap-sort on the heap built from question 1. Explain the time complexity of heap sort? [6 points]

3. Consider the following graph. Perform breadth first traversal and depth first traversal on this graph. In every case, fill out the adjacency list and consider **A** as the beginning vertex. [8 points]



4. Insert the following items into an empty binary search tree in order: {30, 40, 23, 58, 48, 26, 11, 13, 8, 20}. Draw the resulting tree.
a. Write the preorder, inorder and postorder traversals of this tree.
b. What is the maximum height of a binary search tree that contains n items? Why?
c. Write the implementation for finding the minimum element in a binary search tree. You may write either a recursive or iterative implementation.
d. Write the implementation for deleting an element in a binary search tree. You must consider all three cases for your implementation. (5 points)

5. Create a program that provides a framework for experimentation with the variety of sorting algorithms that were covered in class. Using arrays of various sizes, the program should count the number of comparisons and count the number of swaps of each sort. For the purpose of our experimentation, these two counts define our performance criteria. Write a program which inputs an array of integers and sorts them with Bubble sort, Insertion sort, Selection sort, Merge Sort, and Quick Sort. Include counters in your algorithms so you can output how many comparisons and swaps were done by each sorting algorithm, which will indicate the running time of each algorithm.

Run the sorting algorithms on the following lists and verify that the number of comparisons matches the theoretical analysis. For each run, copy the console output and include in your assignment along with the size and type of the array sorted. You should make your program modular enough to allow additional sorts to be added, if required. (You can follow the template code at the end where Insertion sort is already implemented for you, you need to add the remaining sorts)

- 1,000 equal Integers.
- 1,000 random Integers.
- 1,000 increasing Integers.
- 1,000 decreasing Integers.
- 10,000 equal Integers.
- 10,000 random Integers.
- 10,000 increasing Integers.
- 10,000 decreasing Integers.

With the help of Microsoft Excel or other tools, or manually, present your experimental results graphically. Is there a sort that stands out as the best sort based on this performance criteria? Compare your empirical results with the theoretical one for each sorting algorithm (Explain any differences between the empirical and theoretical results, if any). (12 points)

```
#include <iostream>

using namespace std;

static int swaps = 0, comps = 0;

void insertionSort (int arr[], int n)
{
  swaps = 0;
  comps = 0;
  int i, key, j;
  for (i = 1; i < n; i++)
    {
      comps++;
      key = arr[i];
      j = i - 1;

      /* Move elements of arr[0..i-1], that are greater than key, to one
position ahead of their current position */
      while (j >= 0 && arr[j] > key)
        {
          arr[j + 1] = arr[j];
          comps++;
          swaps++;
          j = j - 1;
        }
      arr[j + 1] = key;
```

```
      swaps++;
    }
}

// A utility function to print an array of size n
void printArray (int arr[], int n)
{
  int i;
  for (i = 0; i < n; i++)
    cout << arr[i] << " ";
  cout << endl;
}

// A utility function to print swaps and comparisons
void printSwapComp ()
{
  cout << "Swaps: " << swaps << ", Comparisons: " << comps << endl;
}

int main ()
{
  int size;
  cout << "how big do you want the array?" << endl;
  cin >> size;

  //initialize array
  int array[size];

  //Generate equal element array
  for (int i = 0; i < size; i++)
    {
      array[i] = 1;
    }

  //Generate increasing array
  for (int i = 0; i < size; i++)
    {
      array[i] = i + 1;
    }


  //Generate decreasing array
  for (int i = 0; i < size; i++)
    {
      array[i] = size - i;
    }

  //generate random array
  srand ((unsigned) time (0));
  for (int i = 0; i < size; i++)
    {
      array[i] = (rand () % 100) + 1;
    }
```

```
  //print the array
  printArray (array, size);

  //insertion sort
  insertionSort (array, size);
  printArray (array, size);
  printSwapComp ();

//Bubble sort


//selection sort


//merge sort


  return 0;
}
```