**CSE 101**
**Winter 2022**
**Final Exam**

1. (20 Points) Determine whether the following statements are **True** or **False**. No justification is required.

   a. (2 Points) $n\sqrt{n} = \Omega(n^2)$

      F

   b. (2 Points) $n^{\pi} = O(n^3)$

      F

   c. (2 Points) $n^2 = \Theta\left(9^{\log_3(n)}\right)$

      F

   d. (2 Points) $n\sqrt[3]{n} = \omega(\sqrt{n})$

      T

   e. (2 Points) $n^2 = o(n^3)$

      T

   f. (2 Points) $\ln(n) = o(n)$

      T

   g. (2 Points) $2^n = O(n^2)$

      F

   h. (2 Points) $n^{1.5} = \omega(n^{1.45})$

      T

   i. (2 Points) $n\ln(n) = \Theta(\ln(\ln(n)))$

      F

   j. (2 Points) $f(n) = \omega(f(n))$ for any function $f(n)$

      F

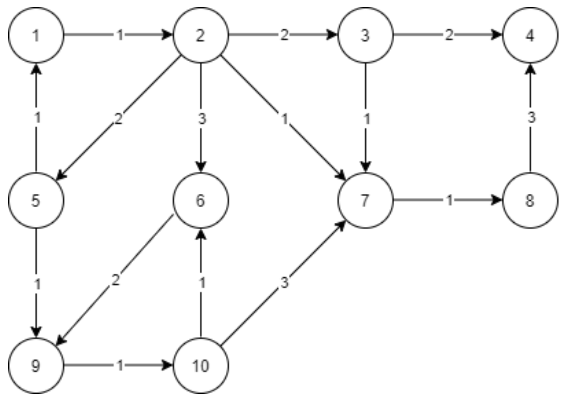
2. (20 Points) Given a Binary Search Tree based on the following C++ struct

```
struct Node{
   int key;
   Node* left;
   Node* right;
};
```

Complete the recursive C++ function below called `TreeWalk()` that takes as input a `Node` pointer `R` and a string `s`, then returns a string consisting of all keys in the subtree rooted at `R`, separated by spaces. The order of the keys depends on the input string `s`, which will be either "`pre`", "`in`" or "`post`", indicating a pre-order, in-order or a post-order tree walk, respectively. If the input `s` is not one of the strings "`pre`", "`in`" or "`post`", then your function will return the empty string.

```
std::string TreeWalk(Node* R, std::string s){
   // your code starts here


  if(R == NIL){
       return "";
  }
  if(s == "pre"){
       return std::to_string(R->key) + TreeWalk(R->left, s) + TreeWalk(R->right, s)
  }
  if(s == "in"){
       return TreeWalk(R->left, s) + std::to_string(R->key) + TreeWalk(R->right, s)

  }
  if(s == "post"){
       return TreeWalk(R->left, s) + TreeWalk(R->right, s) + std::to_string(R->key)

  }
  return "";


   // your code ends here
}
```

3. (20 Points) Perform Dijkstra$(G, s)$ on the weighted digraph below with source vertex $s = 5$. If at some point two vertices have equal minimum d-values, extract the one with smaller label first from the priority queue. (Pseudo-code for Dijkstra can be found here.)




a. (10 Points) Determine the order in which vertices are extracted from the min Priority Queue.

```
extract order: 1 2 7 8 9 10 6 3 4
```

b. (10 Points) For each vertex $x$, determine the values $d[x]$ and $p[x]$.

**Solution:**

| $x$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $d[x]$ | 1 | 2 | 4 | 6 | 0 | 3 | 3 | 4 | 1 | 2 |
| $p[x]$ | 5 | 1 | 2 | 3 | NIL | 10 | 2 | 7 | 5 | 9 |

4. (20 Points) Perform BuildHeap($A$) on the following unordered array, making it into a max-heap. (The heap algorithms can be found here.) Observe that identical keys are accompanied by letters representing different satellite data. Thus the elements 2a and 2b have the same key, but are distinguishable elements in the max-heap.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | 2a | 4 | 7 | 1a | 2b | 3 | 1b | 5a | 2c | 6 | 8 | 5b |

Show the state of array $A$ after the call to BuildHeap($A$).

```
8 6 7 5a 4 5b 1b 1a 2c 2a 2b 3
```

5. (20 Points) Given a connected (undirected) graph $G$, the *diameter* of $G$ is the maximum possible distance between any two vertices $x$ and $y$ in $G$, i.e.

$$\text{diameter}(G) = \max\{\, \delta(x, y) \mid x, y \in V(G) \,\}$$

Using only the Graph ADT functions defined in the project description for pa2, fill in the definition of the client function below that computes and returns the diameter of $G$.

```
int diameter(Graph G){
   // your code begins here

   int max = -1;
   for(int i = 1; i <= getSize(G); i++){
        BFS(G, i);
        for(int j = 1; j <= getSize(G); j++){
             if(getDist(G, j) > max){
                  max = getDist(G, j);
             }
        }
   }
   return max;

   // your code ends here
}
```