lab 4 extra-credit June 4 2% extra credit.

COMPARISON CLASSES

Problem Solving with Computers-II

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

GitHub

std::priority_queue template arguments

```
template <
      class T,
      class Container= vector<T>,
      class Compare = less <T>
      > class priority_queue;
```

type of keys

DS used to implement a PQ.

comparison class.

The template for priority_queue takes 3 arguments:

1. Type elements contained in the queue.
2. Container class used as the internal store for the priority_queue, the default is **vector<T>**
3. Class that provides priority comparisons, the default is **less**

std::priority_queue template arguments

comparison class

```
//Template parameters for a max-heap
priority_queue<int, vector<int>, std::less<int>> pq;
```

max-heap

```
//Template parameters for a min-heap
priority_queue<int, vector<int>, std::greater<int>> pq;
```

min-heap

Comparison class

func'tor

- Comparison class: A class that implements a function operator for comparing objects

greater

```
class compareClass{                  10          20
greater  bool operator()(int& a, int & b) const {
            return a>b;
                   a<b
     }
};
```

Compare Class cmp;

cmp(10,20) // False

priority_queue<int, vector<int>, compareClass> pq

code of Priority Queue

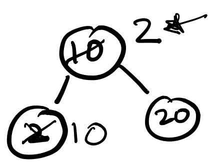
compareClass cmp;

if (cmp (a, b)) {
// a has lower priority than b

else {
// a has high priority than b
}

pq. push (10);
pq. push (2);
pq. push (20);

<u>minHeap</u>



cmp(2,10)
cmp(20, 2)

Comparison class

```
class compareClass{
      bool operator()(int& a, int & b) const {
            return a>b;
      }
};


int main(){
    compareClass cmp;
    cout<<cmp(10, 20)<<endl;
}
```

What is the output of this code?

A. 1

B. 0

C. Error

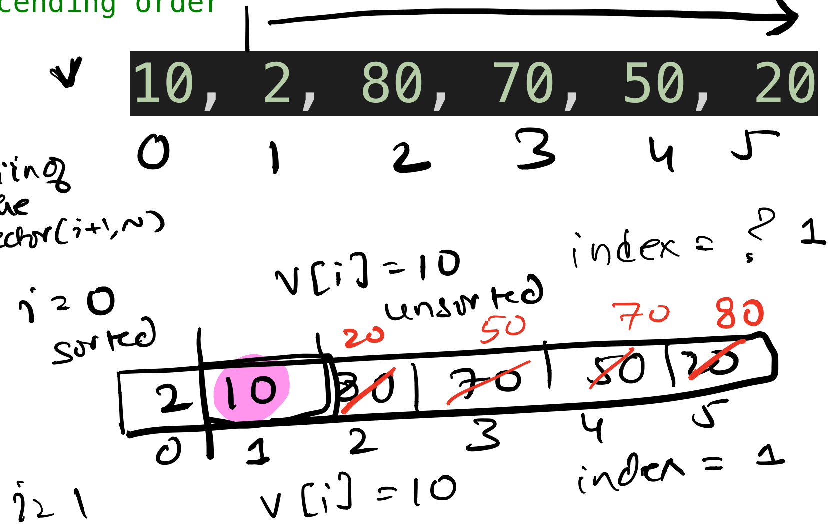
STL Heap implementation: Priority Queues in C++

```
  class comparisonClass{
        bool operator()(int& a, int & b) const {
              return a>b;
        }
  };
priority_queue<int, vector<int>, comparisonClass> pq;
pq.push(10);
pq.push(2);
pq.push(80);
cout<<pq.top();
pq.pop();
cout<<pq.top();
pq.pop();
cout<<pq.top();
pq.pop();
```

pq is a _____heap

Sorting : Selection sort

```
//Precondition: unsorted vector v with N elements
//Post condition: sorted vector in ascending order
void selectionSort(vector<int>& v){
   int N=v.size();
   for(int i = 0; i < N; i++){
      int index=i; 0
      for(int j = i+1; j < N; j++){ min of the vector(i+1,N)
         if(v[j] < v[index]){
            index = j;
         }
      }
      int tmp = v[i];
      v[i] = v[index];
      v[index] = tmp;
   }
}
```

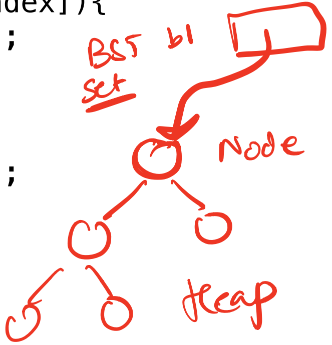

10, 2, 80, 70, 50, 20

Sorting a forest (of Binary Search Trees)

set<int> s0;
BST

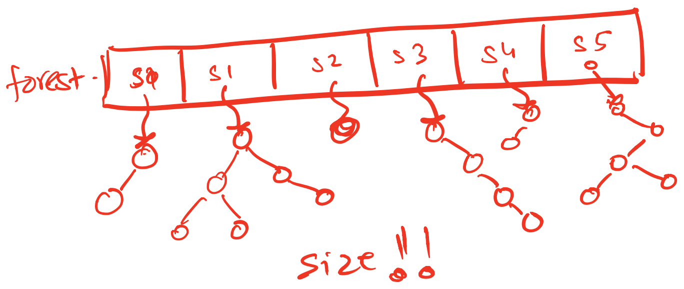
```
//Precondition: unsorted vector v with N elements
//Post condition: sorted vector in ascending order
void selectionSort(vector<int>& v){
    int N=v.size();
    for(int i =0; i< N; i++){
        int index=i;
        for(int j = i+1; j<N;j++){
            if(v[j]<v[index]){
                index = j;
            }
        }
        int tmp = v[i];
        v[i] = v[index];
        v[index]=tmp;
    }
}
```

The idea of a forest of binary trees

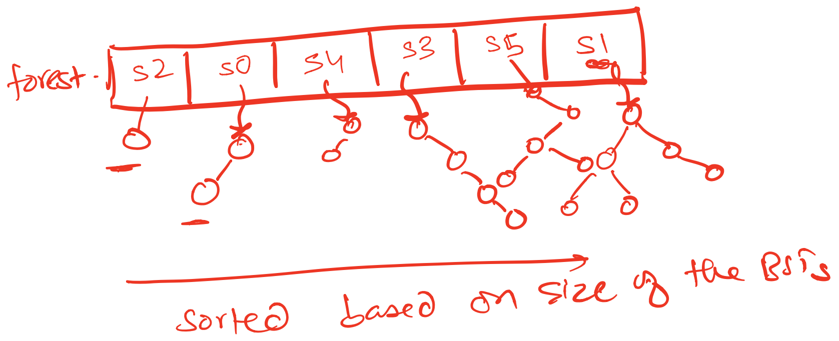


Huffman algorithm

Modify selection sort to work with a forest: vector of bsts



forest
s0
s1
s2
s3
s4
s5
size !!



forest
s2
s0
s4
s3
s5
s1
sorted based on size of the BSTs

Sort array elements using a pq storing pointers

```
int main(){
      int arr[]={10, 2, 80};
      priority_queue<int*> pq;
      for(int i=0; i < 3; i++)
            pq.push(arr+i);

      while(!pq.empty()){
            cout<<*pq.top()<<endl;
           pq.pop();
      }
      return 0;
}
```

How can we change the way pq prioritizes pointers?

Write a comparison class to print the integers in the array in sorted order

```
int main(){
      int arr[]={10, 2, 80};
      priority_queue<int*, vector<int*>, cmpPtr> pq;
      for(int i=0; i < 3; i++)
            pq.push(arr+i);

      while(!pq.empty()){
            cout<<*pq.top()<<endl;
          pq.pop();
      }
      return 0;
}
```