Universidad de Concepción
Facultad de Ingeniería
Departamento de Ing. Informática y Cs
de la Computación

Profesor
Diego Seco

Laboratorio 5:

Estructuras de Datos (503220)

Estudiante: Nicolás Araya (2018448613)

1 y 2:

PriorityQueueHeap:

int top()

```
int PriorityQueueHeap::top(){ //O(1)
    if(empty()) return INT_MIN;
    return _arr.at(1);
}
```

void pop()

```
void PriorityQueueHeap::pop(){     //O(log n)
    if(empty()) return;
    _arr[1]=_arr.back();
    _arr.pop_back();
    downHeap(1);  //O(log n) en peor caso
}
```

```
void PriorityQueueHeap::downHeap(int pos){  //O(log n) en peor caso
    int l = (2*pos);
    int r = (2*pos)+1;
    if(l > size()) return;
    if(l == size()){
        if(_arr.at(pos)>_arr.at(l)){
            swap(_arr[pos], _arr[l]);
        }
        return;
    }
    if(_arr.at(pos)>_arr.at(l) || _arr.at(pos) > _arr.at(r)   ){
        int menor;
        if(_arr.at(l) <= _arr.at(r) ) menor = l;
        else menor = r;
        swap(_arr[pos], _arr[menor]);
        downHeap(menor);
    }
}
```

void push(int)

```
void PriorityQueueHeap::push(int n){   //O(log n)
    _arr.push_back(n);
    upHeap(size());   //O(log n) en peor caso
}
```

```
void PriorityQueueHeap::upHeap(int pos){     //O(log n) en peor caso
    int posUp = (pos)/2;
    if(posUp<1) return;
    if(_arr.at(posUp) > _arr.at(pos)){
        swap(_arr[pos], _arr[posUp]);
        upHeap(posUp);
    }
}
```

int size()

```
int PriorityQueueHeap::size(){  //O(1)
    return _arr.size()-1;
}
```

bool empty()

```
bool PriorityQueueHeap::empty(){  //O(1)
    if(_arr.size()>1) return false;
    else return true;
}
```

PriorityQueueUnsorted:

int top()

```
int PriorityQueueUnsorted::top(){   //O(n)
    if(empty()) return INT_MIN;
    int min = _arr.at(0);
    int menor = 0;
    for(int i = 0; i<_arr.size(); i++){
        if(_arr.at(i)<min){
            min=_arr.at(i);
            menor=i;
        }
    }
    return _arr.at(menor);

}
```

void pop()

```
void PriorityQueueUnsorted::pop(){       //O(n)
    if(empty()) return;
    int min = _arr.at(0);
    int menor = 0;
    for(int i = 0; i<_arr.size(); i++){
        if(_arr.at(i)<min){
            min=_arr.at(i);
            menor=i;
        }
    }
    _arr[menor]=_arr.back();
    _arr.pop_back();

}
```

void push(int)

```
void PriorityQueueUnsorted::push(int n){   //O(1)
    _arr.push_back(n);
}
```

int size()

```
int PriorityQueueUnsorted::size(){ //O(1)
    return _arr.size();
}
```

bool empty()

```
bool PriorityQueueUnsorted::empty(){  //O(1)
    return _arr.empty();
}
```

3)

Heap Sort:

```
void HeapSort(vector<int> &vec){  //O(n logn)
    PriorityQueueHeap* pqh = new PriorityQueueHeap();
    for(int i = 0; i<vec.size(); i++) pqh->push(vec.at(i));
    vec.clear();


    while(!pqh->empty()) {
        vec.push_back(pqh->top());
        pqh->pop();
    }
}
```

Selection Sort:

```
void SelectionSort(vector<int> &vec){ //O(n^2)
    PriorityQueueUnsorted* pqu = new PriorityQueueUnsorted();
    for(int i = 0; i<vec.size(); i++) pqu->push(vec.at(i));
    vec.clear();
    while(!pqu->empty()) {
        vec.push_back(pqu->top());
        pqu->pop();
    }
}
```