# California State University, Fresno

# DEPARTMENT OF COMPUTER SCIENCE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **Algorithms & Data Structures** | | | Semester: | **Spring 2022** |
|  | | | | | |
| Points |  | Document author: | **MingkuanPang** | | |
|  | Author’s email: | **Yafking20 @mail.fresnostate.edu. email** | | |
| Laboratory number: | **03** | | |
|  | | | | | |

**1. Statement of Objectives**

This lab asks for implementation of Priority Queue ADT using Binary Heaps. There are implementations of the operations of Priority Queue ADT in this lab.

**2. Experimental Procedure**

**Maximum Heap:**

**modifyValMax:**

A screenshot of a computer

Description automatically generated with medium confidence.

For this procedure, I changed the argument “int Arr[]” to vector “vector<int>Arr”. Because vector is easier to operate. To keep the properties of maximum heap., when modify a value in the array, compare the new value with its parent, if it is greater than its parents, then exchange their values. Repeat this until it gets the first element of the array.

**Insert\_value\_maxHeap:**

Text

Description automatically generated with medium confidence

After adding new element to the maximum heap, use max\_heapify function on the last not-leaf node, because insertion starts at the bottom of the heap, and from left to the right.

**max\_heapify**

Text

Description automatically generated

In the max\_heapify function, **int l** is the index of child node on the left side, **int r** is the index of child node on the right side. Make the argument element to compare with its left child and its right child, eventually if any of these two children is the largest one, then switch the value of the argument element to that child. Finally, make the largest child to be the argument for the next round.

**extract\_maximum**

Text

Description automatically generated

First use buildMaxHeap to make the array to be a maximum heap, then the first element will be the maximum element in the array. After assigning its value to a value holder, switch its value to the value of the last element, then put the first element (it is the last element of the original array.) into max\_heapify to keep the properties of maximum heap.

**ascendingHeapSort**

A screenshot of a computer

Description automatically generated with medium confidence

First, make the array to be a maximum heap, then switch the value of first element to its last element. After switching, use max\_heapify to keep the properties of maximum heap. Therefore, the largest element will be throwed to the back of the array. Keep doing this until it reaches the first element of the array.

**buildMaxHeap:**

A screenshot of a computer

Description automatically generated with medium confidence

To build a maximum heap, only the nodes that is not leaf will be necessary to process, which will be the range of Arr [0… (n/2)]. Therefore, make every node in the range of Arr [0… (n/2)] to be the argument of max\_heapify to make the whole array to have the properties of maximum heap.

**Minimum Heap:**

**ModifyValMin**

Text

Description automatically generated

The basic idea is the same as modifyValMax. To keep the properties of minimum heap., when modify a value in the array, compare the new value with its parent, if it is smaller than its parents, then exchange their values. Repeat this until it gets the first element of the array.

**Min\_heapify**

Text

Description automatically generated

In the min\_heapify function, **int l** is the index of child node on the left side, **int r** is the index of child node on the right side. Make the argument element to compare with its left child and its right child, eventually if any of these two children is the smallest one, then switch the value of the argument element to that child. Finally, make the smallest child to be the argument for the next round.

**Insert\_value\_minHeap:**

Text

Description automatically generated

After adding new element to the minimum heap, then start with the last node of the heap, make the comparison with its parents, if its parent is greater than it, then switch its value with its parents.

**extract\_minimum**

Text

Description automatically generated

First use buildMinHeap to make the array to be a minimum heap, then the first element will be the minimum element in the array. After assigning its value to a value holder, switch its value to the value of the last element, then put the first element (it is the last element of the original array.) into mini\_heapify to keep the properties of minimum heap.

**descendingHeapSort**

A screenshot of a computer

Description automatically generated with medium confidence

First, make the array to be a minimum heap, then switch the value of first element to its last element. After switching, use min\_heapify to keep the properties of minimum heap. Therefore, the largest element will be throwed to the back of the array. Keep doing this until it reaches the first element of the array.

**buildMinHeap**

A screenshot of a computer

Description automatically generated with medium confidence

To build a minimum heap, only the nodes that is not leaf will be necessary to process, which will be the range of Arr [0… (n/2)]. Therefore, make every node in the range of Arr [0… (n/2)] to be the argument of min\_heapify to make the whole array to have the properties of minimum heap.

**Create Vector**

Text

Description automatically generated

Create vector reads a string, and sperate the element(number) in the string by blank, and put those elements in an array, then returns the array as the result.

**3. Analysis**

**Output:**

Text

Description automatically generated

The program asks user to make a choice first, (0 for maximum heap, 1 for minimum heap), after user making a choice, then create\_vector() will be called, and it will ask user to input a list of numbers then it will make those numbers to be an array. Then insert\_value\_max\_heap() will be called(for choice 0, and insert\_value\_min\_heap() will be called for choice 1), it will ask user for a number, and insert the number into the array. After that, ascending\_heap\_sort will be called (for choice 0, and decending\_heap\_sort will be called for choice 1). Then the current content of the array will be printed on the screen.

After that, extract\_maximum () will be called (for choice 0, and extract\_minimum() will be called for choice 1). It will remove the maximum (for choice 0, and minimum for choice 1) number of the array and then print it out. After that, ascending\_heap\_sort will be called again (for choice 0, and decending\_heap\_sort will be called for choice 1), then the current content of the array will be printed out.

**4. Encountered Problems**

The argument of the procedures was “int Arr []” which was fixed-length array before. That was hard for me to do the insertion and deletion. Therefore, I changed it to be vector. The length of a vector can change freely.

**5. Conclusions**

This lab took me to analyze how to change an array into a maximum heap or a minimum heap, which made me more proficient in various operations of heap and deepened my understanding of heap.

**6. References**

I did not use any references in this report.