A heap or priority queue

Nariman Haidar

October 2022

1 Introduction:

In this assignment we are going to learn about a heap, or priority queue. First we are going to make the simplest way of implementing a priority queue. Then we are going to make a heap with queue linked list and then with array. Then compare between them and se the complexity time to each of them.

2 Method:

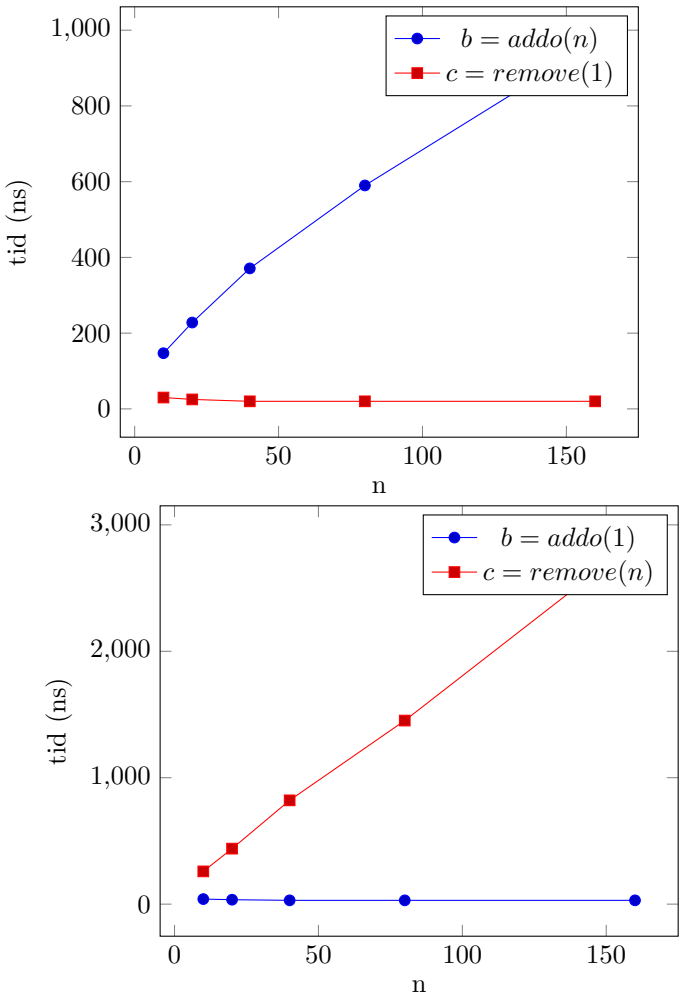
A list of items:

One of the ways that we implement a queue with is linked list. In this assignment we need to implement queue and de queue methods by given the priority to the lowest value. Here we are going to implement this in two different ways. First we are going to make the queue method based on their priority. Then in the other way we are going to make de queue that remove the element with the highest priority. To implement the code I started by make a class of node and the class of queue list and one time I make the add method that take a constant time and I make a remove method that remove the least element in this queue and it takes o(n) time complexity because it going to go through the whole list looking for the least element so it takes o(n). I make some of bench mark to measure the time. Then I switch the case and I make the add take time complexity o(n) and that to add an element in order for example if we have a list of elements 3,4,6,9 and 11 and we want to add 8 so it is going to through the list until it finds the 9 and then it will set the 8 before the 9 element. Then I make remove method that is so simple so it just remove the first element that we add it and it takes a constant time. Then I do some bench mark to measure time.

I think that when the add is taking o(n) and remove is taking o(1) its better than that opposite because when the add it doesn't going through the whole list in the average case when it find the element that is greater than the new element so it just set the new element before that but in the case and when we add at from the beginning of the list we make some compare to set the items sorted then the remove method it will just remove the element that was sorting,

but when add take constant and remove doesn't so it should go through the whole list to find the min element because it is not sorted when we make add all elements. We can see the difference from the graph.

Graph:queue o(n) and de-queueo(1)



balancing the tree:

A heap is a tree data structure and can be classified as a complete binary tree. All the nodes of the heap are arranged in a specific order. Heap property is the relationship between the root and his children. There is two types of the heap first type is max heap and that is the root is the greatest key of all other keys in the heap. The other type of heap is min heap and here the root node has key

that is the smallest or minimum among all the other keys in the heap.

In our assignment we are going to write a heap binary tree which is the same of binary tree but here in our case we have is to keep the tree balanced and root has priority. And In the The add method we are going to use recursively so every time a new element is added will all steps above to be repeated.So to implement this code I started with at create a class heap tree and then class node that have priority, size, left and right then I make a constructor to the class node. Then I started with add method that take integer prio as a parameter then if the prio is smaller than the priority so swap then if the left null set the new node in the left and if the right is null so set the new node in the right then if the right size is smaller than the left size so add the prio to the right +1 else set the prio to the left. In the add method the time complexity time is o(logn) because it going t add an element and the it going to compare between elements then swap. The remove method is a little complicated more than add method because we need to consider all cases.To implement remove method first it going to check if the left is null so it return the right and check if the right is null so it return the left then if the left priority is smaller then the right priority so remove the left else remove the right and then sink the size with one. In the remove method the time complexity is o(logn). Then I write the method print to print out all the nodes. Then I make the en queue method and that is if the root is not null so just add else the root is the new node prio Then I make the method de queue that if the root is not null set the ret is the root priority and then remove the root else return null. Then I make som bench mark to measure the time. The time complexity to the heap tree is o(logn).

increment a element:

To now we implement the heap we are going to see how much is it balanced by seeing the depth of the tree. First we are going to add 64 random elements to the tree then we will to push random value to the end of the tree which then it will returns deep. Then we are going to collect some statistics on the depth that the push operation needs to go down. In the add method we are going to the bottom of the tree and then find the depth and then put the the value but if it is not in the right place so it will compare between values and then put it in the right place. The push method it going to use the root and then push the values down. Then write some benchmark to collect statistics on the add method.

table

An array implementation:

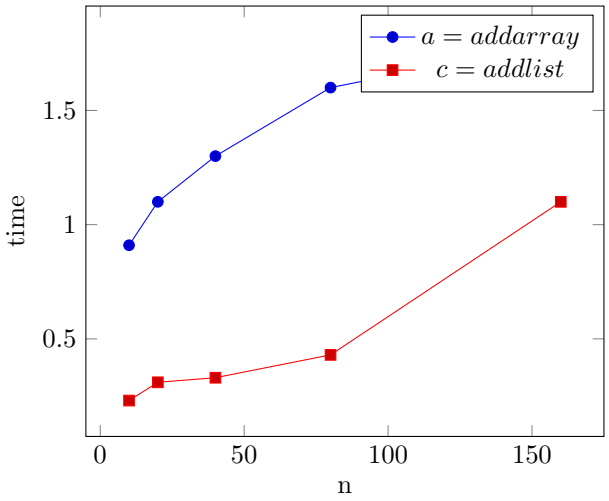
In this section of the assignment we are goin to write a heap using array and as it is a complete binary tree, it can easily be represented as an array. First the root element will be A[0] where A is the array used to represent the binary heap, and to represents the parent node it going to be A [(i-1)/2] and the left

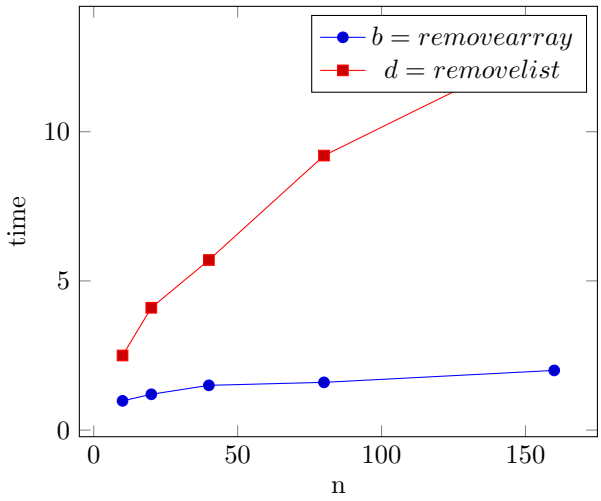
Increment	add	push
10	3	2
13	3	4
15	6	4
16	4	6
20	4	6
21	4	6
22	5	6
23	5	6

Table 1: This table to know depth of add and push

child is A[(2*i)+1] and the right child is A[(2*i)+2]. To implement the heap with array I start to writ methods that define the parent and left child and right child.In add method we are going to add element in the root position and at the same time all elements going to be placed last free position in vector. To keep the tree balanced with the elements we are going to compare each element with its parent if the elements are smaller As they age, they change places. In the remove method we are going to remove the root and then we are going to compare witch child is smaller then we swap. Then I make some bench mark to measure the time. Time complexity to the heap using array is o(logn).

Compare with the time implementation to the heap as a linked structure and array:




$b = removearray$
$d = removelist$
time
n
0
5
10
0
50
100
150