An In-Depth Analysis of Heap Data Structure

Muhammad Hassan, Muhammad Sameed Abbasi, Fariya Khan, Nimrah Altaf Adam, Fasih-ur-Rehman Ansari

Department of Software Engineering

NED University of Engineering and Technology, Karachi, Pakistan

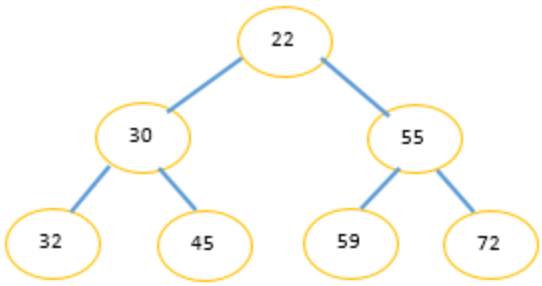
Abstract*— *A heap is a very fundamental and important data structure used in various application softwares as well as in system software structure that provides a well-organized implementation of priority queue operations. Numerical Data often has to be sorted for its submission in different contexts and different fields. The data available is sorted through different algorithms based on various factors like time, efficiency, complexity, and other etc. This research report emphases on binary heap data structure, its various techniques, operations and merits and de-merits.it further describes the innovative and purely functional implementation method of binary heaps. A binary heap is a tree-based data structure that implements priority queue operations like insert, delete, extract minimum or maximum element, merge/meld; that assure at worst logarithmic running time for them. The information discussed in this paper presents a simple and in-depth analysis of binary heap data structure.

Keywords*— *Binary Heaps, Data Structures, Priority Queues, Time Complexity

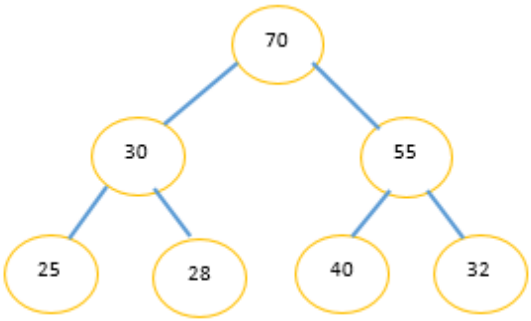
I. INTRODUCTION

A heap is a special kind of data structure, also known as a binary heap that can be constructed using an array, which is based on tree data structure and fulfills the property of heap i.e. the key linked with the parent node, is either higher than the key of the child nodes or lesser than the key of the child nodes. Therefore, the heap in which the key of the parent node is greater than the key of the child nodes, in a heap structure is acknowledged as the max heap, and if the key of the parent node is lesser than the key of the child nodes, throughout the heap structure, it is acknowledged as min heap. Heap data structure has extensive usage like in heapsort procedure, in several graph algorithms, as well as to efficiently signify dynamic priority lists. A good aspect of heap structure is its competence to be efficiently characterized in arrays without making the use of explicit pointers. It uses array data type to execute its operations and Its array-based form supports the operations in worst-case time, and insert in worst-case time using at most element comparisons. Additionally, its pointer-based method supports delete and decrease in worst-case time via at most element comparisons.[1] Heap categorized into two main types; Maxheap, in which elements with a higher value have a higher priority and the other is Minheap, in which elements with a lesser value have a higher priority. The basic operations of a (min-)heap are: build-heap, heapify, insert, find-minimum, and delete-minimum. A Heap is a Binary Tree Structure that is either a tree is Full or can say a Complete Binary Tree or the key value of each individual node is greater than or equal to the key value in each of its children; or we can say it follow two properties Ordering and Structural. In **ordering**, Nodes must be arranged in an order according to values. The values should follow min-heap or max-heap property.[2]

In **min-heap** property, the value of each node is greater than or equal to its parent value, with the minimum value at the root node.



In **max-heap** property, the value of each node is fewer/less or equal to its parent value, with the maximum value at the root node.

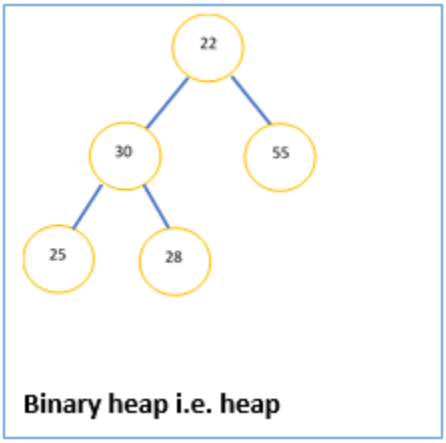


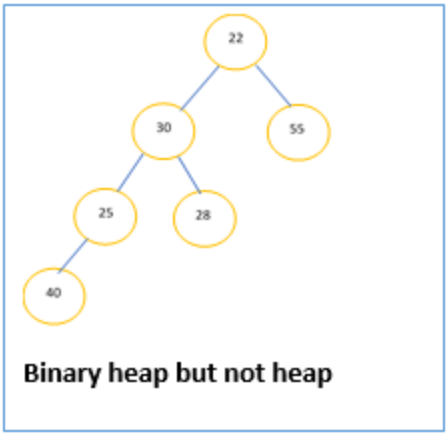
And in **STRUCTURAL ,** All levels in a heap should be full i.e. it is called a complete binary tree.

. All phases of heap should be full, excluding the last one.

· Nodes must be occupied from left to right strictly.

Heap does not follow Binary Search tree rules so the values in right and left nodes does not matter.





II. BACKGROUND

In computer science priority queues are considered one of the most effective methods to solve problems related to sorting for example the single-source shortest-paths problem, and the minimum-spanning-tree problem etc. Heap data structure play a very significant role in solving these problems due to the positioning of the biggest and smallest element at the root of the tree for a max-heap and a min-heap respectively. Since heaps are controlled by swapping nodes, this allows for in-place sorting and hence makes it useful for priority queues.The weak heap is a priority queue that was introduced as a competitive structure for sorting. In this paper we study the application and properties of weak-heap data structure that distinguish it from an ordinary binary heap. One of the topics covered is the implementation of Dijkstra's algorithm to prove the efficiency of weak heaps over other data structures. The performance of weak heaps is tested through multiple experiments by considering different set of data elements and observing the actual running time.[3]

This paper talks about the birth of binary heaps and its evolution over time on which research has been done. Much work has been done on design and analysis of priority queues. The binary heap data structure was created to store a dynamic set of data elements from an ordered set which supported the operation to insert an element and to extract the minimum element. The comparisons by the earlier

model were too much so research was done to reduce them which resulted in the formation of d-ary heaps. Over time, research introduced the world with double-ended priority queues and then with self-adjusting priority queues. Various operations were also introduced such as EXTRACT-MAX, MELD, DECREASE-KEY, etc. The complexities of the existing operations reduced as research on heaps progressed. This paper gave an overview of some of the advancements and many more branches have not been discussed here.[4]

In O(log n) worst case time, a binary heap is known for supporting insert and extract-min, where n denotes the number of elements contained in the data structure. We implement an in-place binary heaps variant that supports insert in O(1) time, and extract-min in O(log n) time, all in the amortized sense, with comparisons of elements at most log n + O(1).

In O(1) worst-case time, we demonstrate how to support insert and extract-min as effectively as binary heaps at the same time. We demonstrate how a series of extract-min operations can be carried out in-place after linear preprocessing using at most log n + O(1) element comparisons per operation. In our view, it is interesting that by marginally loosening the restrictions that are inherent to the lower limits, we might circumvent two lower limits, thought to be solid for a long time. We developed an algorithm to build an in-place structure in linear time, which would then support minimum extraction in the worst-case time of O(log n) and comparisons of elements per operation at most log n + O(1).[5]

A priority queue, a data structure that supports the minimum (top), insert (push) and extract-min (pop) operations, among other things, is said to operate in-place if it uses extra space O(1) in addition to the n elements stored at the beginning of an array. No in-place priority queue was known prior to this work to provide worst-case guarantees on the number of element comparisons that are ideal for both insert and extract-min up to additive constant terms. Insert and extract-min operate in logarithmic time for the standard implementation of binary heaps, while involving at most lg n and 2 lg n [could probably be limited to lg lg n O(1) and lg n log O(1)] element comparisons, respectively. It is impressive that by slightly loosening the assumptions that are intrinsic to these lower limits, we could surpass the lower limits on the number of element comparisons performed by insert and extract-min accepted for binary heaps.[6]

This topic is a glimpse into a new efficient heap operation called a binary heap. A binary heap is similar to a binary search tree (Okasaki, 1999). with two new accessors operations to new fields in a heap - its height and size. This additional data should be accessible in constant time to define efficient and simple search criteria for insert and remove operations. Both insertion and removal are followed by construction, thus constantly altering the size of the heap, therefore the size and height of the heap play an important role in an efficient binary heap.[7]

III. OPERATIONS AND THEIR RUN-TIME COMPLEXITIES

There are many heap operations that can be carried out on this data structure but this paper will look deeply into some of those while mentioning the other ones.

Find-max/min:

The operation **find-max** returns the maximum item in a max-heap and **find-min** returns the minimum item in a min-heap, also called the root element. The time complexity of this operation is **O(1)** for max-heap and min-heap, both.

Extract-max/min:

Extract-max is another operation that removes the maximum node from max-heap, returns the value, and restructures the remaining nodes to make it a max-heap. The time complexity of this algorithm is **O(log n)** as after popping the root node, the tree needs to maintain the property of the heap.

Extract-min has the same function as Extract-max but it applies to a min-heap and has the same time complexity of **O(log n)**.

Heapify:

A process to rearrange the elements of the heap in order to maintain the heap property is called heapify. It is done when a certain node disturbs the heap properties in the heap due to some operation on that node.

Heapify can be done in two methodologies:

up_heapify() → It follows the bottom-up approach. In this, we check if the nodes are following heap property by going up towards the root node and if nodes are not following the heap property we do certain operations to let the tree follow the heap property.

down_heapify() → It follows the top-down approach. In this, we check if the nodes are following heap property by going down towards the leaf nodes and if nodes are not following the heap property we do certain operations to let the tree follow the heap property.

```
Procedure heapify(Hp[], index, n) {
/* n : number of nodes, Hp [1..n]: heap of size n,
   index: index of a tree node */
1.    pos  = 2 * index;
2.    item = Hp[index];
3.    not_done = true;
4.    while (pos < n and not_done) do
5.        if (pos < n – 1 and HP[pos] > Hp[pos+1])
6.            pos++;
7.        if (item <= Hp[pos])
8.            not_done = false;
9.        else  Hp[pos/2] = Hp[pos];
10.       pos = 2*pos;
11.   end while
12.   Hp[pos/2] = item;
13.   }
```

Figure 1.0[8]

Insertion:

Insertion in Min-Max Heap requires you to create a node first. Then the value to be inserted in the heap is stored and then swapping is carried out to heapify the structure if required. The following figures show how exactly does insertion work and what changes are done in the array in the background.

Consider Figure 1. 1 which shows the current state of the heap as a tree and as an array and node 67 which needs to be inserted.

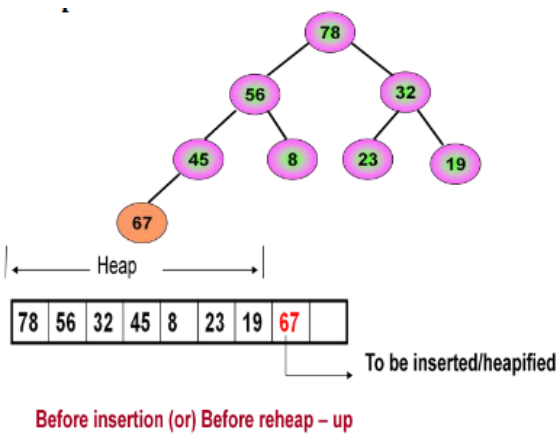


Figure 1. 1

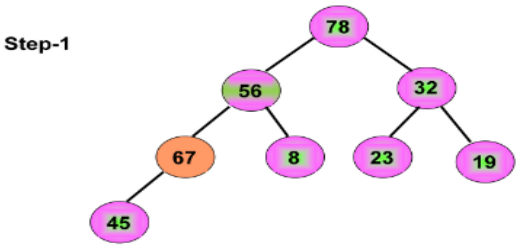


Figure 1. 2

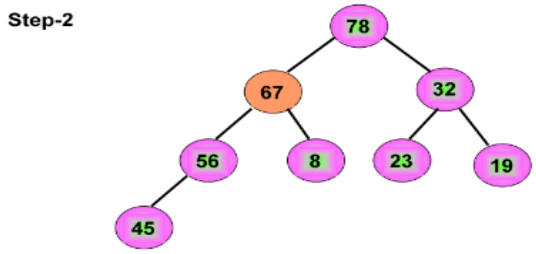


Figure 1. 3

Step-3 As the node 67 has been placed in its correct position so the array will look like this.

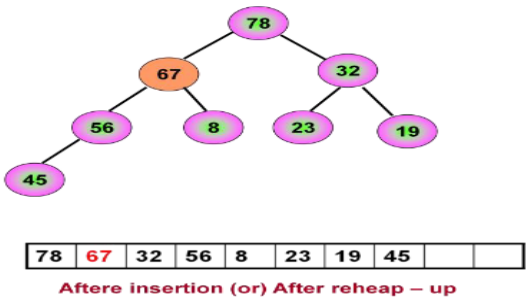


Figure 1. 4

A sample algorithm to carry out insertion is,

Insert a node containing the value to insert in the left-most location of the lowest level of the Binary Tree.[9]

Move the inserted node up using this algorithm:

While (inserted node's value < its parent node's value)

{

Swap the values of these two nodes;

}[9]

The time complexity from this algorithm and the figures shown above can be clearly stated to be **O(log n)** because of the swaps required to maintain the heap property.

Deletion:

Deletion in Min-Max Heap is an operation that deletes the required node from the heap and restructures the broken heap to satisfy the heap properties again.

Consider Figure 2. 1 which shows the present state of the heap before the deletion of any node. The node to be deleted is highlighted and that is node 78, the root node.

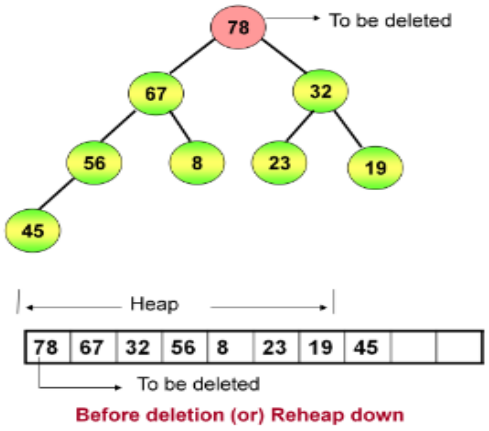


Figure 2. 1

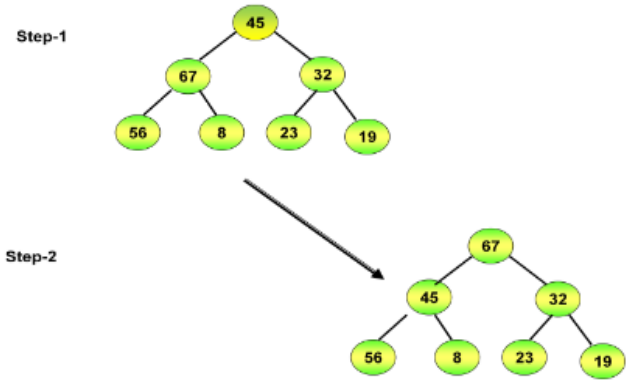


Figure 2. 2

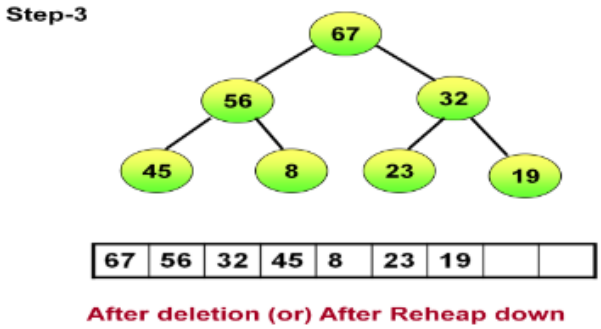


Figure 2. 3

A sample algorithm to carry out deletion in a heap is as follows,

1. Delete a node from the array

(This results in a space being vacant and the tree is no longer complete)

2. Replace the deletion node with the rightmost node on the lowest level of the Binary Tree (This makes it into a complete binary tree)

3. Heapify:

If (value in replacement node < its parent node) then,

Shift the replacement node "up" the binary tree.

Else,

Shift the replacement node "down" the binary tree

As the above figures show the deletion of the root node so this is considered as the worst-case scenario of this operation as all of the nodes would need to be

restructured to make it a heap again. Therefore, the time complexity of this particular case is **O(log n)**. However, if we consider it generally for all nodes so a representation of its complexity in Big Theta would be better which is **Θ(log n)**.

Meld:

Another important operation related to heap data structures is meld which is also known as merge. This operation joins two heaps to make a new valid heap containing all the elements of the two heaps while deleting or destroying the previous original heaps.

The following is a pictorial representation of meld operation.

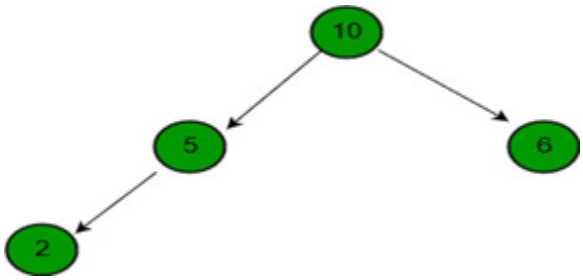


Figure 3. 1[10]

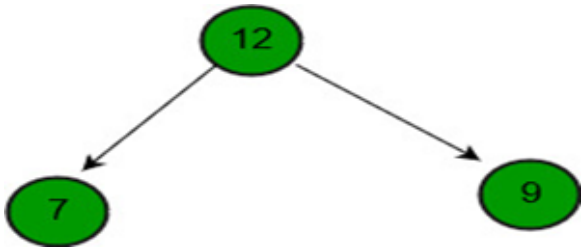


Figure 3. 2[10]

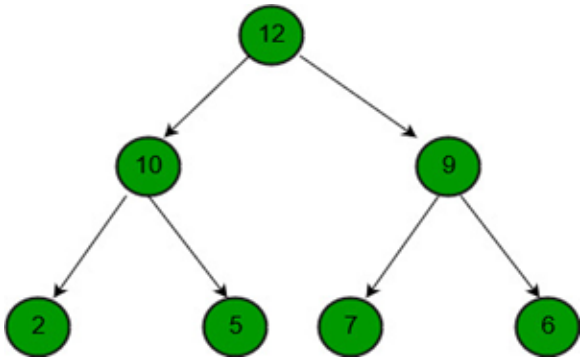


Figure 3. 3[10]

Two max heaps need to be merged as shown above. The following is a possible algorithm to achieve what is in Figure 3. 3[10],

Step-1 Create an array to store the output of this algorithm

Step-2 Copy the elements of both the arrays/heaps one by one to the result array

Step-3 Build a heap to construct a valid merged max-heap.

Upon analysis of the algorithm, the time complexity can be determined. As the time complexity to build the heap from an array of **n** elements is **O(n)**. Therefore, the complexity of melding/merging the two heaps is equal to **O(m+n)**.

Some other operations related to heap data structures include **heapify** which, upon giving an array of elements, creates a heap. **Increase-key** or **Decrease-key** updates a key within a max or min-heap, respectively. **Sift-up** and **Sift-down** move the node up a tree or down the tree to restore the properties of the heap, respectively.

IV. APPLICATIONS OF HEAP

Heap Data Structure is usually taught with Heapsort. Heapsort algorithm has restricted utilizations on the grounds that Quicksort is better practically speaking. However, the Heap data structure itself is widely used. Following are some uses other than Heapsort.

Priority Queues

Efficient implementation of Priority queues can be performed using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in O(logn) time complexity. Some variations of Binary Heap are Binomoial Heap and Fibonacci Heap. The union of these variations takes O(logn) time which is an O(n) operation in Binary Heap. Graph algorithms like Prim's Algorithm and Dijkstra's algorithm are one of the common Implementations of priority queues.[11]
Order statistics: The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array.

Huffman coding

Huffman coding could be a successful lossless compression method used originally for text

compression. Huffman's idea is, rather than employing a fixed-length code like 8 bit extended ASCII or DBCDIC for every symbol, to represent a frequently occurring character in an exceedingly source with a shorter codeword and to represent a less frequently occurring one with an extended codeword.

Algorithm: Huffman encoding

Step 1: Build a binary tree where the leaves of the tree are the symbols within the alphabet.

Step 2: the perimeters of the tree are labeled by a 0 or I.

Step 3: extract the Huffman code from the Huffman tree.

INPUT: a sorted array of one-node binary trees (t1, t2, t3, . . ., tn) for alphabet

(S1, S2, . . ., Sn) with frequencies (W1, W2, . . . ,Wn),

OUTPUT: a Huffman code with n code words.

Algorithm: Huffman decoding

Step 1: Read the coded message one bit at a time. Ranging from the basis, we traverse one edge down the tree to a toddler per the bit value. If this bit read are 0 moves to the left child, otherwise, to the correct child.

Step 2: Repeat this process until a leaf is reached. If a leaf is reached, decode one character and restart the traversal from the basis.

Step 3: Repeat this read-and-move procedure until the top of the message.

INPUT: a Huffman tree and a 0-1 (binary) bit string of encoded message

OUTPUT: decoded string.

Dijkstra's algorithm

Dijkstra's algorithm is extremely like Prim's algorithm for minimum spanning tree. Like Prim's MST, we create a shortest path tree known as SPT with the given source as root. We keep up two sets, one set contains vertices remembered for the shortest-path tree, the other set incorporates vertices not yet a part of the shortest-path tree. At each progression of the algorithm, we find a vertex which is inside the other set (which is not yet included) and encompasses a minimum distance from the starting source.

Below are the detailed steps utilized in Dijkstra's algorithm to seek out the shortest path from one source vertex to any or all other vertices within the given graph.[12]

Algorithm

1) Create a set of the shortest path tree (sptset) that keeps a record of vertices included in shortest path tree, i.e., whose shortest distance from the starting source is calculated and concluded. Initially, this set is an empty/null set.

2) Assign a distance value to any or all vertices within the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex in order that it's picked first.

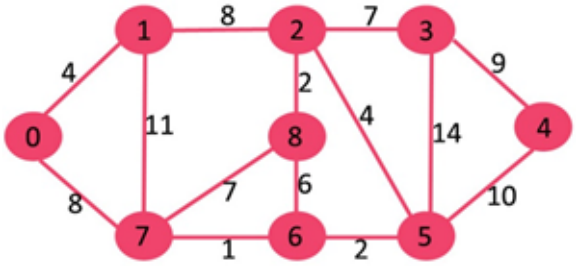
3) While sptSet doesn't include all vertices

….a) Pick a vertex u which isn't there in sptSet and has the smallest distance value.

….b) Include u to sptSet.

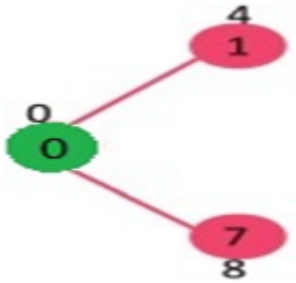
….c) then the distance value of all adjacent vertices of u is updated. To update the gap values, iterate through all adjacent vertices. for each adjacent vertex v, if the sum of distance value of u (from source) and weight of edge u-v, is a smaller amount than the space value of v, then update the space value of v.

Let us understand with the subsequent example:



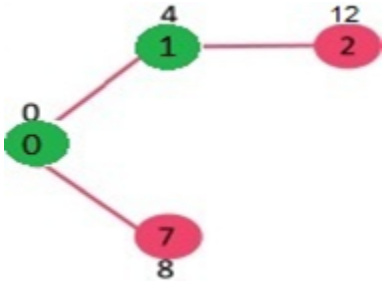
The set sptSet is initially a null set and distances allotted to vertices are where INF indicates infinite. Now pick the vertex with the smallest distance value. The vertex 0 is chosen and included in sptSet. Now sptSet becomes {0}. After 0 is included to sptSet, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and seven. The space values of 1 and seven are updated as 4 and eight. Vertices and their distance values are shown in the following subgraph, only the finite distance value

vertices are shown. The vertices that are included in SPT are represented in green color.



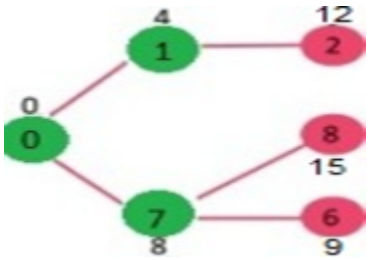
Pick the vertex with the smallest distance value and which is not already included in SPT. The vertex 1 is picked and inserted to sptSet. So sptSet now changes into {0, 1}.

Update the gap values of adjacent vertices of 1. The gap value of vertex 2 becomes 12.



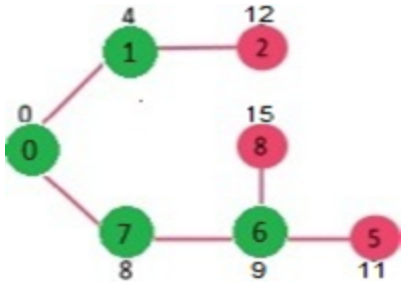
Pick the vertex with the smallest distance value and not already included in the SPT set. Vertex 7 is picked. So sptSet now changes into {0, 1, 7}.

Update the gap values of adjacent vertices of seven. the space value of vertex 6 and eight becomes finite (15 and 9 respectively).

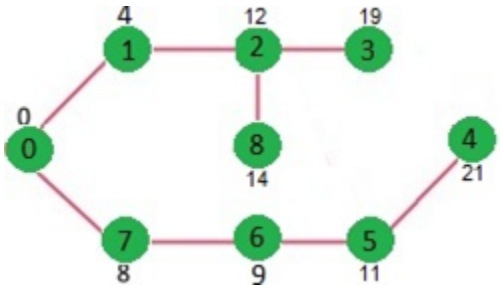


Pick the vertex with the smallest distance value and not already included in the SPT set. Vertex 6 is picked. So sptSet now changes into {0, 1, 7, 6}.

Update the gap values of adjacent vertices of 6. The space value of vertex 5 and eight are updated.



We repeat the above steps until sptSet does include all vertices of the given graph. Finally, we get the subsequent Shortest Path Tree (SPT).



V. ADVANTAGES AND DISADVANTAGES

In Computer Science Sorting a bunch of elements in an array is a task that occurs frequently. A sorting algorithm is a technique that can be utilized to put a list of unordered elements into an arranged succession. A key is generally used to control the grouping of elements. There is a large variety of sorting techniques available which differ on the basis of responsiveness and performance from one another. The Heap sort algorithm is generally utilized due to its proficiency. It is by far one of the finest sorting methods for being in-place and with no quadratic worst-case scenarios. Most programming languages use heap to store global variables. Heaps back up the dynamic allocation of memory. The heap is not run mechanically instead it is more like a free-floating region of memory managed manually by the CPU.[13][14]

Heaps are remarkable for executing a priority queue due to the positioning of the biggest and smallest item at the root of the tree for a max-heap and a min-heap respectively. A min-heap is used for a min-priority queue and a max-heap is used for a max-priority queue. If you imagine a heap as a binary tree stored sequentially in a hierarchical

manner, with the basic node first (then the children of that node next, then the children of those nodes next and so on); then the children of a node at index N are at 2N+1 and 2N+2. This function then permits brisk access-by-index. And since heaps are controlled by swapping nodes, this allows for in-place sorting and hence makes it useful for priority queues.[18][19]

Efficiency

The Heap sort Algorithm is proficient. With an increment in number of items heap sort increases logarithmically while on the other hand sorting algorithms like Quick Sort etc. generally tend to become exponentially slower. This suggests that Heap sort is especially useful for sorting a list containing a large number of elements. The execution of Heap sort is quite ideal and this implies that no other sorting algorithms can perform better in contrast.[14] For example if we want to find the greatest and least value or the k-th largest element it can be found in linear time (often steady time) using heaps. It is basically a combination of time efficiency of merge sort and the storage efficiency of quicksort. [15]

However one of the main issues of heap sort is memory fragmentation as the heap space is not used efficiently. Memory allocation and deallocation is performed manually by the programmer. The memory is allocated in a random order, therefore, it needs explicit memory deallocation. As a result the memory becomes fragmented. Where as in data structures like stack, memory is located in a continuous block and this allows Space to be managed efficiently by Operating System so memory will never become fragmented.[17]

Memory Usage

The Heap sort algorithm can be executed as an in-place sorting algorithm. This implies that its memory usage is minimum because apart from what is necessary to hold the primary list of elements to be sorted, it needs no extra memory space to run.[14] When heap sorting is performed on objects that do not have any reference, Garbage Collection is run on the heap memory to release it. All the objects generated in the heap space have global access and can be called from any part of the application.[16]

Conversely, both the Merge sort and quick sort algorithm require a much larger memory space due to the recursive nature of the algorithms. Other than that stack is dependent upon the operating system for its space size whereas when it comes to heap sort there is no specific size limit on memory space.

Memory management is more complex in Heaps because it is used globally. Therefore the Heap memory is divided into Young-Generation and Old-Generation. In Applications where there is a requirement to allocate a large section of memory for storing data like for example, you want to generate a large size array or build big frameworks to hold the variables around for a long time then memory should be allocated using heap data structure. However, if you are working with comparatively smaller variables that are only needed until the operations using them are actively running, then in such a situation using a heap data algorithm is not wise as it is slower and more complicated.

Simplicity

The Heap sort algorithm is easier to understand than other equally proficient sorting algorithms. Since it does not make use of intermediate computer science concepts as in recursion, it becomes easier for programmers to run their systems correctly. It is much more flexible as compared to other data structures as it allows resizing of variables.[14]

Although the algorithm may be simpler but when it comes to accessibility and locality of reference a heap data structure is much slower as compared to data structures like stack.[17]

As heaps generally use an array-based data structure instead of pointers, the execution of the operations tend to be faster as compared to a binary tree which makes use of pointers. Also merging of complicated heaps like binomial is easier than integration of other sorting algorithms.[15]

One of the function of a heap is that it is a framework that maintains partially ordered data sets; thus, it is an adequate tradeoff between the expense of maintaining a complete order and the cost of going through random chaos.[18]

Consistency

The Heap sort algorithm exhibits steady execution. This implies it runs equally well in the best, average and worst cases. As a result of its ensured execution, we can say that it is appropriate to use heaps in applications with a critical response time.[17] In order to test the performance of a heap sort algorithm counts of compare and exchange operations were

made for three different sorting algorithms running on the same data:[20]

n	Heap Sort		Quick Sort		Insert Sort	
	Comparison	Exchange	Comparison	Exchange	Comparison	Exchange
100	2,842	581	712	148	2,595	899
200	9,736	1,366	1,682	328	10,307	3,503
500	53,113	4,042	5,102	919	62,746	21,083

The table shows that if n is large then using a heap sort algorithm is the better choice because of its guaranteed O (nlog n) time. When it comes to life-critical (medical emergency, life support in airship and spaceship etc.) and mission-critical (handling hazardous matter in industrial and research plants etc.) software, they will generally have a response time as a feature of the system framework. In such systems, it is not appropriate to configure based on an average performance algorithm like quick sort.

In cases where n is small it is better to use insertion sort algorithm as it has simpler code and therefore smaller steady factors. Also it is much more stable when handling small sets of data as unlike heap sort it does not change the relative order of elements with equal keys.

Moreover, Heap Sort Algorithm takes more time to compute due to which most commercial applications would prefer to use quicksort for its better average performance. They can put up with a periodic long run (which implies that a report takes slightly more time to produce on full moon days in leap years) in exchange for shorter runs most of the time.[20]

VI. CONCLUSION

In this paper we generally discussed the binary heap data structure, its related operations and its pros and cons with their applications. Binary heaps are mostly used in building priority queues and heap sort algorithm. There are two significant steps to the heap sort algorithm. The first important step includes the conversion of the entire tree into a heap. The second important step is to remove the largest portion from the root and turn the remaining tree into a heap. Heap sort has guaranteed O(n*log(n)) efficiency, although the constant factor is substantially higher than other algorithms, such as quick sorting. Unlike the other n log n types, the memory efficiency of the heap type is constant, O(1), since the algorithm for heap sort is not recursive. Heap sort is not a stable kind, so it is not possible to preserve the initial ordering of equal elements.

VII. REFERENCES

[1]https://kclpure.kcl.ac.uk/portal/files/87388857/TheoryComputingSzstems.pdf

[2]https://study.com/academy/lesson/heap-data-structure-definition-properties.html

[3]https://www.sciencedirect.com/science/article/pii/S1570866712000792

[4]https://pure.au.dk/portal/files/72054050/C313.pdf

[5]http://cphstl.dk/Report/Binary-heaps/cphstl-report-2013-1.pdf

[6]https://kclpure.kcl.ac.uk/portal/files/87388857/TheoryComputingSzstems.pdf

[7]https://arxiv.org/pdf/1312.4666.pdf

[8]http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.119.7807&rep=rep1&type=pdf

[9]http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/9-BinTree/heap-insert.html

[10]https://www.geeksforgeeks.org/merge-two-binary-max-heaps/

[11]International Journal of applied science Trends and Technology (IJCST) – Volume 5 Issue 1, Jan – Feb 2017 ISSN: 2347-8578 www.ijcstjournal.org Page 61

[12]www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/– 30-9-2020– page 7

[13]https://www.sciencedirect.com/science/article/pii/S1570866712000792

[14]https://sciencing.com/the-advantages-disadvantages-of-sorting-algorithms-12749529.html

[15]http://z-sword.blogspot.com/2014/02/advantages-and-disadvantages-of-sorting.html

[16]https://stackoverflow.com/questions/5227976/what-is-the-use-of-the-heap-data-structure

[17]https://www.guru99.com/stack-vs-heap.html

[18]https://stackoverflow.com/questions/749199/when-would-i-want-to-use-a-heap

[19]https://www.codesdope.com/blog/article/priority-queue-using-heap/#:~:text=Priority%20queues%20ar

e%20used%20in,and%20a%20min%2Dheap%20respectively.

[20]https://www.cs.auckland.ac.nz/software/AlgAnim/qsort3.html#:~:text=Data%20Structures%20and%20Algorithms%3A%20Quick%20vs%20Heap%20Sort&text=Empirical%20studies%20show%20that%20generally%20quick%20sort%20is%20considerably%20faster%20than%20heapsort.

VIII. BIOGRAPHY

Muhammad Hassan (SE-19058):

I, Muhammad Hassan, am currently an undergraduate student of Software Engineering at NED University of Engineering and Technology. I have done O and A Levels from Beaconhouse School System and have a Computer Science background of five years. Apart from my interest in this field, one major thing that made me pursue it is the fact that it is constantly emerging and expanding which would bring greater job opportunities for me in the near future. I have plans to enter in the domain of Cyber security or Artificial Intelligence mainly because I love coding and mathematics and hacking but also because of the vacant seats in these areas I can foresee.

Muhammad Sameed Abbasi (SE-19059):

My self, Muhammad Sameed Abbasi, currently studying Software engineering at Ned University of engineering and technology, karachi, Pakistan. Having interest in maths and physics I choose the field software engineering as nowadays the young generation get attached towards technology and is currently a rising field with a broad scope. As per my experience I do not have any clear interests and am still exploring the domain. Knowing that this field is ever growing, so I'll surely give my best to become a successful software engineer.

Fariya Khan (SE-19066):

I, Fariya Khan, a software engineer student at NED University of Engineering and Technology. Completed my intermediate from PECHS College having the background of computer science. Having no interest in computer engineering, I unexpectedly find myself in software engineering not because I desired to but because of this progressive field evolving day by day spreading its roots to nearly every field of life. Now finding this field completely different from my interest, I am struggling my level best to accomplish what I want to, learning new things, and discovering new strategies will definitely benefit the world. I will be looking forward toward graphic designing from the aspect of software engineering, creating user friendly interfaces for innovative data visualization to reduce complexity to greater extent.

Nimrah Altaf Adam (SE-19077):

My name is Nimrah Altaf Adam. I am a Software Engineer student at NED University. I have done both my O-levels and A-levels with computer science as one of my subjects. So I had extensive knowledge regarding this field before starting my four year degree in Software Engineering. There were multiple reasons due to which I opted for this field. Nowadays freelancing and online jobs are growing more and more as it is providing people with opportunities to take up part time or full time jobs and work from home according to their own convenience. This field will help me develop such skills that will allow me to explore this domain more. Also during my teenage years, I once read an article about 3D printers and robots. This was the first time I came across the concept of Artificial Intelligence. Since then I have been intrigued and fascinated with Superintelligence. For now Web Development and Artificial Intelligence are the two fields I am interested in and I am working towards my progress in these two domains.

Fasih-ur-Rehman Ansari (SE-19090):

My self Fasih-ur-rehman Ansari, currently the student of Software engineering at Ned University. Being an avid lover of e-sport gaming I choose the field software as nowadays the young generation get attached towards technology , entirely every field is now being advanced and becoming more closer towards machines and robotics. As per my interest I planned to be involved in the gaming domain. Knowing that this field do-not have a downfall right now so I'll be surely giving my best to become a successful software engineer.