INFO 6205 – Program Structures and Algorithms
Crash Course in Algorithms – Written Section

Student Name: Vidhi Bharat Patel - 002762999
Professor: Nik Bear Brown

Heap Sort Algorithm

Definition

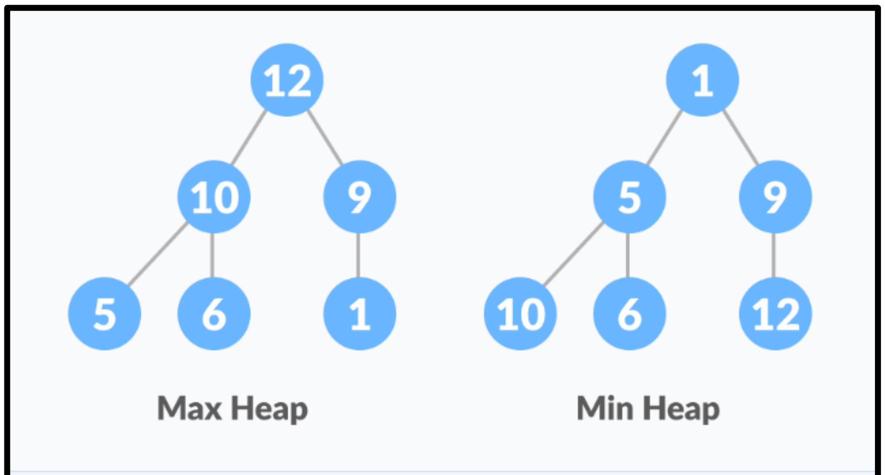
What is a heap?
A complete binary tree is referred to as a heap, and a binary tree is a tree in which a node can have a maximum of two offspring. A complete binary tree is one in which all nodes are left-justified and all levels, with the exception of the last level, the leaf node, are fully filled. (upGrad, 2020)

What is heap sort?
Among sorting algorithms, heapsort is well-liked and effective. Heap sorting is the process of removing each element from the list's heap portion one at a time and adding them to the list's sorted portion. The in-place sorting technique used is called heapsort.

Heap sort is basically a comparison-based sorting technique which is based on Binary Heap data structure. It is similar to the selection sort which is when we first find the minimum element or number and place the minimum element or number at the beginning. This process is repeated for the remaining elements. The typical implementation of Heap sort is not stable but it can be made stable. Heap sort is 2 to 3 times slower than the QuickSort algorithm; it is due to the lack of locality of reference. (GeeksforGeeks., 2022)

Binary Heap

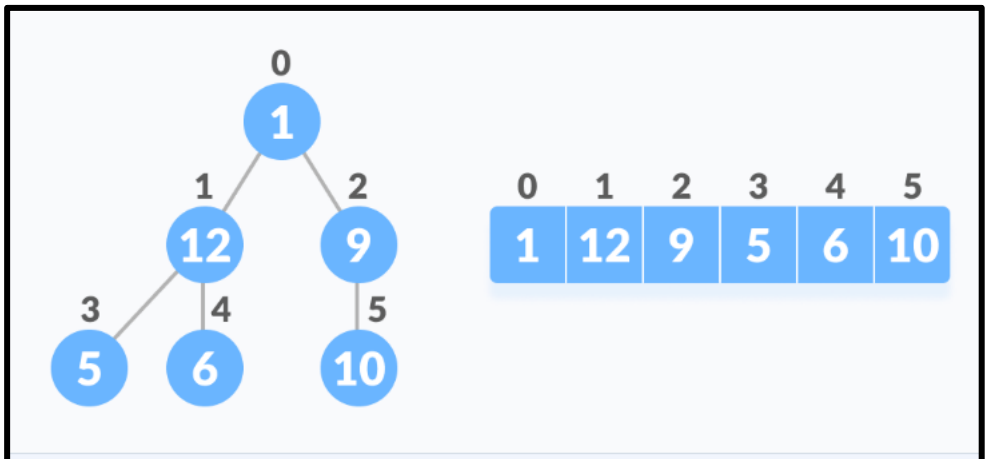
In a binary heap, each parent node is greater than or equal to each of its offspring, essentially creating a binary tree. The heap can be visualized as an array with the root element at index 0 and the left and right children of each element i at index 2i + 1 and index 2i + 2, respectively.
Binary heaps come in two different varieties: max heap and min heap. The root of a max heap contains the highest element possible, whereas the root of a min heap contains the lowest element possible.



Max Heap and Min Heap

Heap Building
The first step in creating a heap from an array is to arrange its components in a binary tree structure. In order to maintain the heap property, we can achieve this by iterating through the array from middle to beginning and comparing each element with its offspring before swapping them out as needed. This procedure is repeated for each element in the array.

Sorting the Array
We first create a max heap from the array before sorting the array using a heap. The initial member, which is the maximum element, is then switched with the last element as we cycle through the array from end to beginning, bringing the heap size down by 1. The remaining elements are then re-heapified in order to keep the heap property, and we continue in this manner until the entire array is sorted.



Relationship between array and heap indices

What is meant by Heapify?
Heapify is the process of converting a binary tree into a Heap data structure. A binary tree is a data structure in the shape of a tree, with all nodes located as far to the left of one another as possible and all levels filled, except for the last. Every level of the binary tree, with the exception of the lowest one, should be filled in for a heap to be considered complete. At this stage, it fills from left to right. The value stored at each node must be more important than or equal to the value stored at its progeny in order for a heap to satisfy the heap-order property. (Academy, n.d.)

Algorithm for Heapify:
```
heapify(array)
 Root = array[0]
   Largest = largest( array[0] , array [2 * 0 + 1]/ array[2 * 0 + 2])
if(Root != Largest)
 Swap(Root, Largest)
```

Heapify is used to first turn the array into a heap data structure, after which the root node of the Max-heap is sequentially deleted and replaced with the last node in the heap, and finally the root of the heap is heapified. Continue doing this until the size of the heap exceeds 1.

From the input array provided, create a heap.
Up until there is just one element left in the heap, repeat the following steps:

1. Replace the heap's root element, which is also its largest element, with its last member.
2. Take out the last component of the heap, which is now in the proper place.
3. Heapify the pile's remaining components.

Reversing the order of the elements in the input array results in the sorted array.

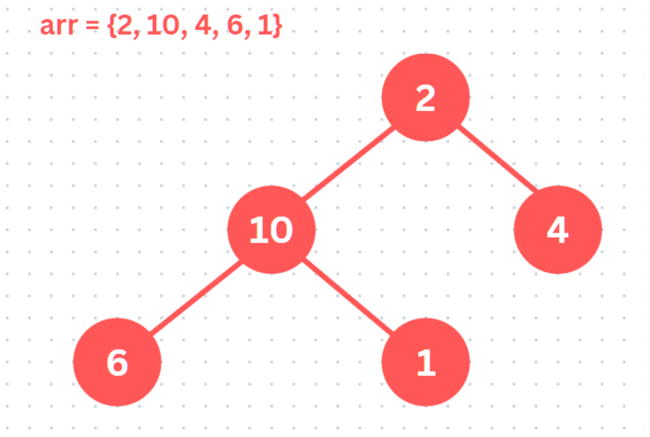
To solve any Heap sort algorithm problem, it is best to follow the below steps:-

- Create a max heap using the provided information.
- At this time, the heap's root contains the maximum element. The last item in the heap should be used in its position, and then the heap should be shrunk by one. Finally, heapify the tree's root.
- As long as the heap is larger than 1, repeat step 2.

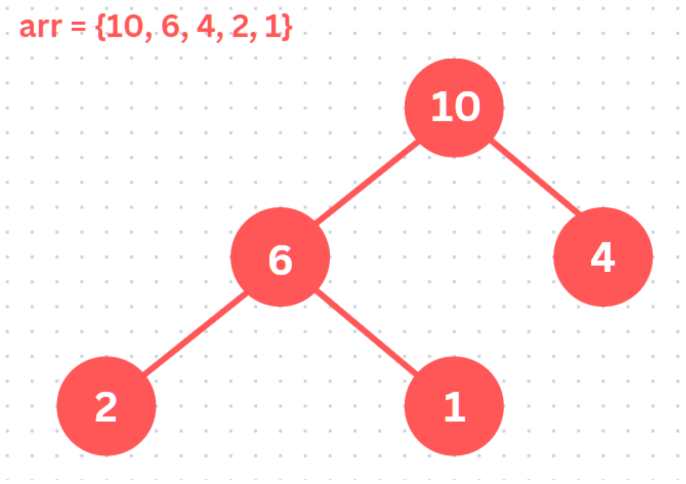
Working of Heap Sort Algorithm

To understand the heap sort algorithm more clearly, we can start off with an unsorted array and try to sort it using heap sort. Consider the array: arr[]= {2, 10, 4, 6, 1}.

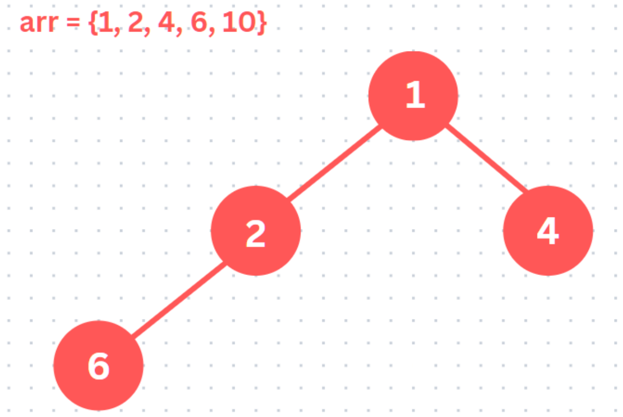
1. Build Complete Binary Tree from the array

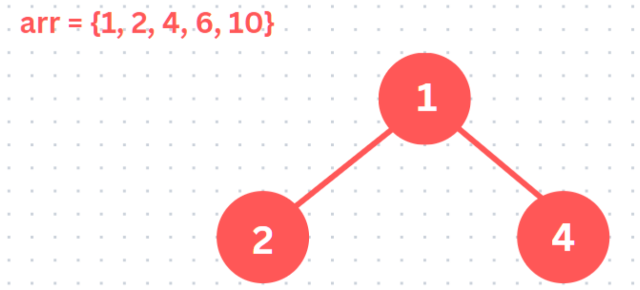


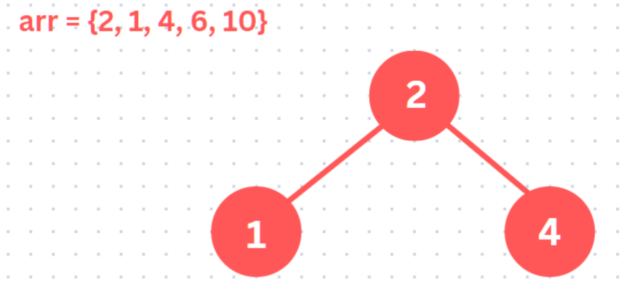
2. Transform into max heap: The task is to create a tree from this unsorted array and try to convert it into max heap.
   - In the example, the parent node 2 is smaller than the child node 10, therefore, it should be swapped to build a max heap.
   - Next, 2 is smaller than the child 4, therefore it should also be swapped and the resulted array should look like this:



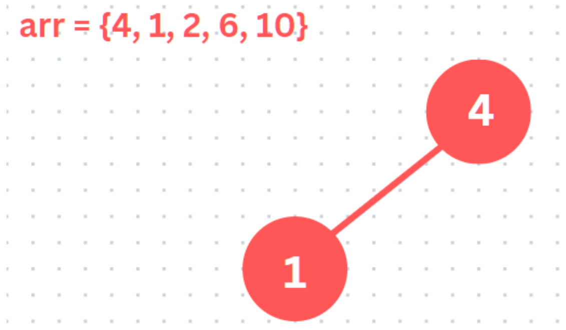
3. Perform heap sort: Remove the maximum element in each step and then consider the remaining elements and transform it into a max heap.
   - Delete the root element (10) from the max heap. In order to delete this node, try to swap it with the last node, i.e. (1). After removing the root element, again heapify it to convert it into max heap.
   - The resulted array looks like:



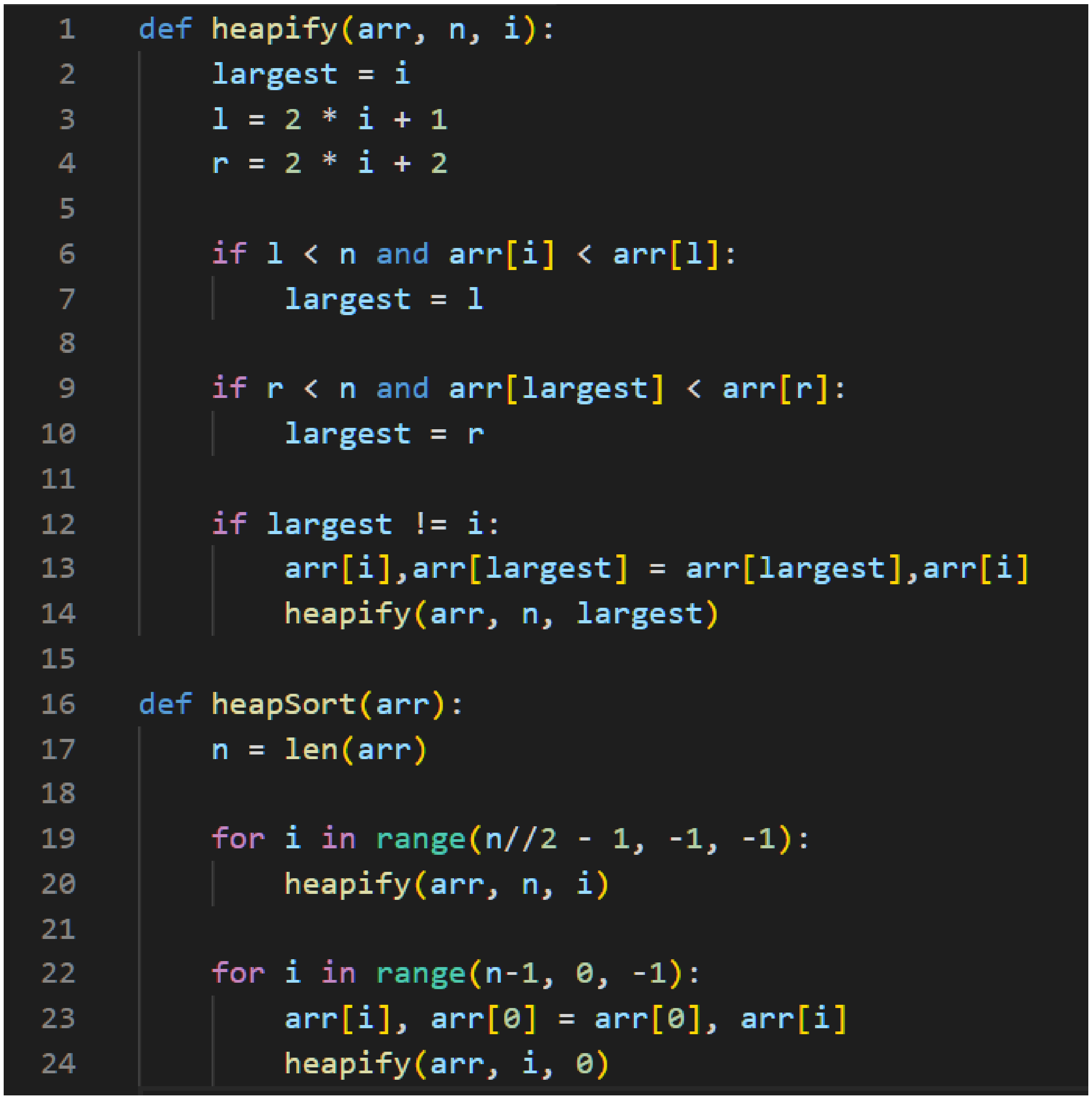




4. Repeat the steps and it will result as followed:
   - Next, remove the root (i.e. 4) again and perform heapify.
   - When the root is removed once again, it will be sorted. The sorted array will be **arr[] = {1,2,4,6,10}.**

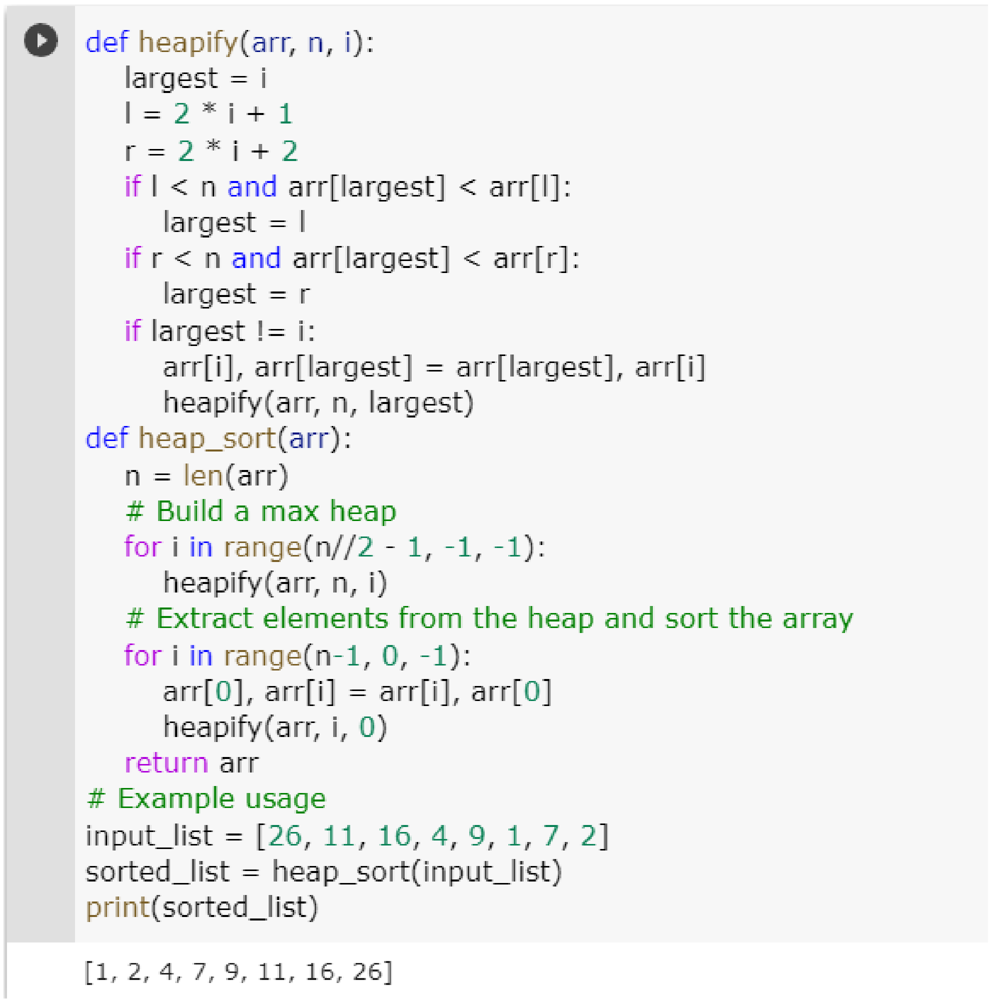


Implementing Heap Sort Algorithm
The following is the psuedo code for Heap sort algorithm:


```
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[i] < arr[l]:
        largest = l

    if r < n and arr[largest] < arr[r]:
        largest = r

    if largest != i:
        arr[i],arr[largest] = arr[largest],arr[i]
        heapify(arr, n, largest)

def heapSort(arr):
    n = len(arr)

    for i in range(n//2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
```

In this method, an array is turned into a maximum heap using the heapify function, and the maximum heap is used to sort the array using the heapSort function. For each non-leaf node in the array, the heapify function is used to create the maximum heap. Up until the entire array is sorted, the elements are taken from the max heap and sorted one at a time.

Following is the Python code for Heap sort algorithm:



```python
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and arr[largest] < arr[l]:
        largest = l
    if r < n and arr[largest] < arr[r]:
        largest = r
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)
def heap_sort(arr):
    n = len(arr)
    # Build a max heap
    for i in range(n//2 - 1, -1, -1):
        heapify(arr, n, i)
    # Extract elements from the heap and sort the array
    for i in range(n-1, 0, -1):
        arr[0], arr[i] = arr[i], arr[0]
        heapify(arr, i, 0)
    return arr
# Example usage
input_list = [26, 11, 16, 4, 9, 1, 7, 2]
sorted_list = heap_sort(input_list)
print(sorted_list)
```

```
[1, 2, 4, 7, 9, 11, 16, 26]
```

Heap Sort Complexity

Time Complexity	
Best	**O (n log n)**
Worst	**O (n log n)**
Average	**O (n log n)**
Space Complexity	**O (1)**
Stability	**No**

Heap Sort has O(nlog n) time complexities for all the cases (best case, average case, and worst case). This makes it an efficient sorting algorithm, especially for large datasets. However, it requires extra space to store the heap data structure, which can be a disadvantage in some scenarios.

Applications of Heap Sort algorithm

1. **Sorting Large Data Sets**
   Heap sort is very useful for sorting large data sets in memory-constrained environments, where the size of the data exceeds the available memory. It can sort large data sets efficiently by dividing them into smaller pieces, sorting them in memory, and merging them together.

   Example: If we have a large dataset of student records, containing millions of records, and we want to sort them based on the student's grade point average (GPA) in ascending order. Heap Sort can be used to sort the data efficiently by dividing the dataset into smaller pieces, sorting them in memory, and merging them together.

2. **Priority Queue**
   A priority queue is a type of data structure that can support the operations insert and extract-max (or extract-min) while storing a collection of elements. Utilizing a heap data structure, Heap Sort can be utilized to efficiently create a priority queue.

   Example: Suppose you have a messaging app that needs to deliver messages to users based on their priority, where higher priority messages need to be delivered first. Heap Sort can be used to implement a priority queue efficiently by using a heap data structure to store the messages, where each message has a priority value associated with it.

3. **Operating Systems**
   For managing memory and scheduling activities, heap sort is a common operating system component. Processes can be sorted according to priority, and sorting the free memory blocks in a heap can be utilized to efficiently allocate memory.

   Example: Suppose you have an operating system that needs to schedule processes based on their priority, where higher priority processes need to be executed first. Heap Sort can be used to sort the processes based on their priority, and to allocate memory efficiently by sorting the free memory blocks in a heap.

4. **Computer Networks**
   The shortest path between two nodes in a network is found via heap sort in network routing algorithms. It can be used to effectively find the shortest way by sorting the routing tables according to the distance to the target node. (Cormen, Leiserson, Rivest, & Stein)

   Example: Imagine that you have a network of routers that must identify the quickest route between two network nodes. In order to efficiently extract the shortest path, heap sort can be used in network routing algorithms to order the routing tables according to the distance to the destination node.

5. **Database Systems**
   Large data sets are sorted using a technique called heap sort in database systems before joining and aggregating the data. By reading and sorting only a small piece of the data at a time, it can be used to efficiently sort the data stored on disk.

   Example: Imagine you have a database system that needs to organize huge amounts of data before joining and aggregating them. By reading and sorting just a small piece of the data at a time, Heap Sort can be utilized to efficiently organize the data on disk.

References

1. Academy, K. (n.d.). *Unit: Algorithms*. Retrieved from https://www.khanacademy.org/computing/computer-science/algorithms

2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (n.d.). Introduction to Algorithms.

3. GeeksforGeeks. (2022). *Heap Sort Algorithm*. Retrieved from https://www.geeksforgeeks.org/heap-sort/

4. Programiz. (n.d.). Retrieved from Data structures and algorithms, Heap sort: https://www.programiz.com/dsa/heap-sort

5. upGrad. (2020, November 23). *Heap Sort in Data Structures: Usability and Performance*. Retrieved from https://www.upgrad.com/blog/heap-sort-in-data-structures/