Heapsort

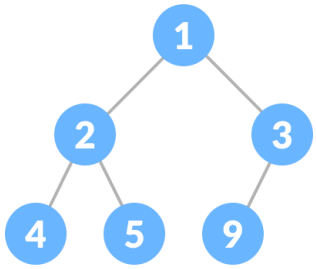
CS 5002 Group: Sylvia, Louise, Shirley, Wenqian

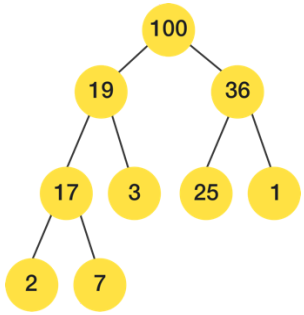
Introduction

Understanding Heap Sort:

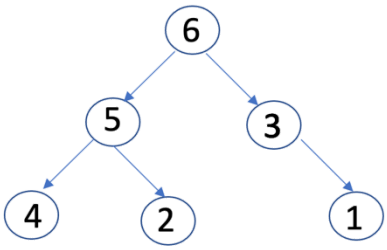
I. What is Heap?

Heap is a complete binary tree, which means all except the last level of nodes are completely filled, and the lowest level is filled to a certain point from the left. The image on the lower left is a complete binary tree where the node containing "3" in the last second level only has 1 child "9". The image on the lower right is also a complete binary tree where the node contains "17" in the last second level that has 2 children "2" and "7".

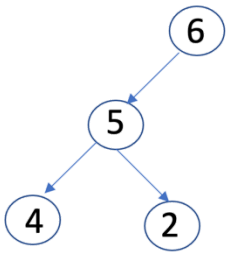




In addition to the correct heap example, below are two incorrect examples of heaps.

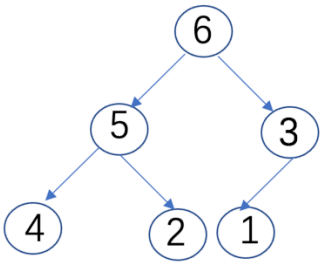


Not Heap

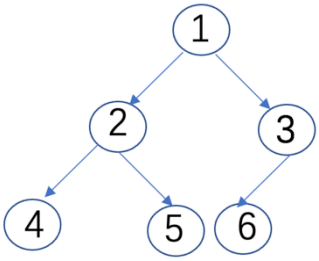


Not Heap

The first type of heap is called max heap, in which all parent nodes are greater than or equal to the values of their children, subsequently the root node contains the greatest value in the heap. The other type of heap is called min heap, in which all parent nodes are less than or equal to the values of their children, subsequently the root node contains the least value in the heap.



Max Heap



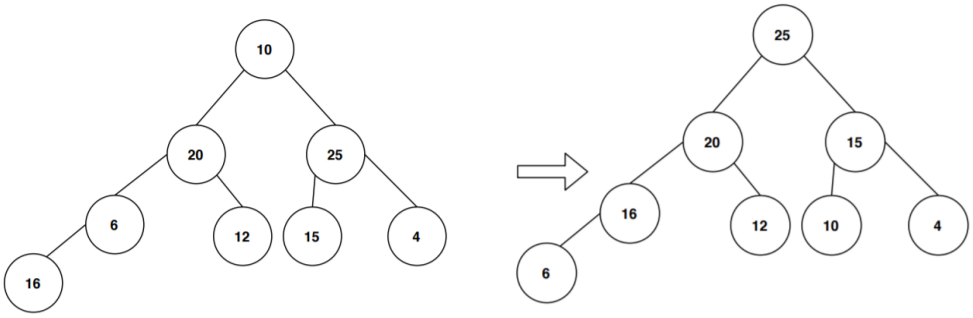
Min Heap

II. What is heapify?

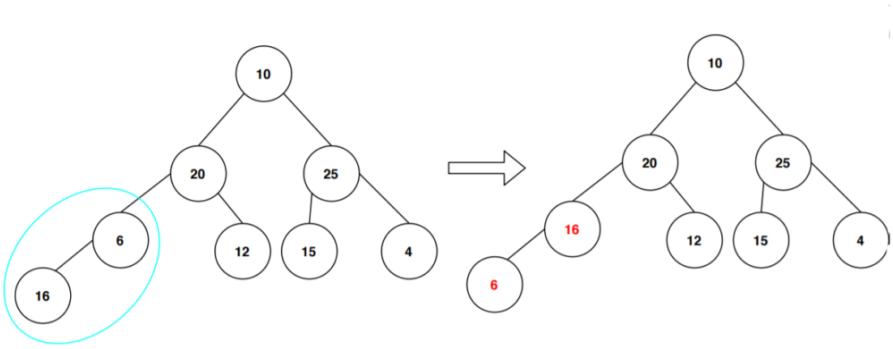
Heapify is the process of creating a heap from a given array or a binary tree. In other words, heapify is the process to rearrange the elements to maintain the property of heap data structure. Just like heap, there are also two types of heapify.

The first type of heapify is called max-heapify. As its name, max-heapify arranges the nodes in correct order so that they follow the property of max heap, in which each parent node is greater than its children and the root node contains the greatest value in the heap.

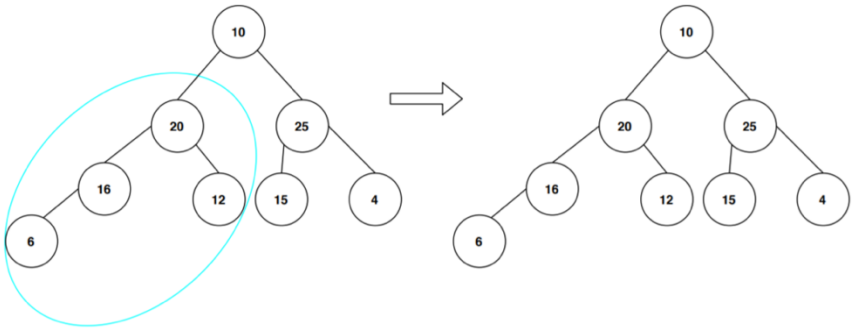
The image on the lower left is the binary tree that doesn't follow max heap's property and the image on the lower right is the binary tree that has been max-heapified.



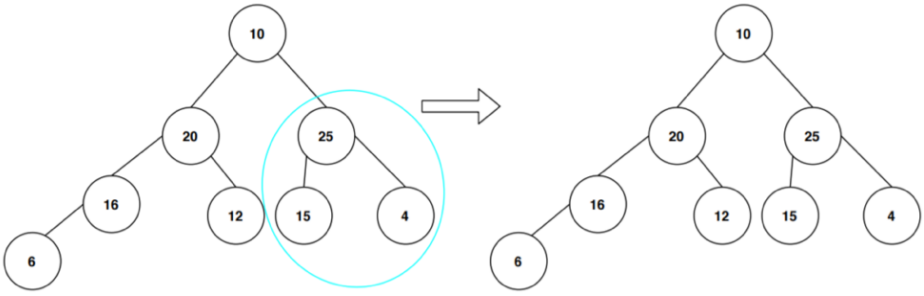
Here are the steps we can follow to swap the nodes and reach max heap.



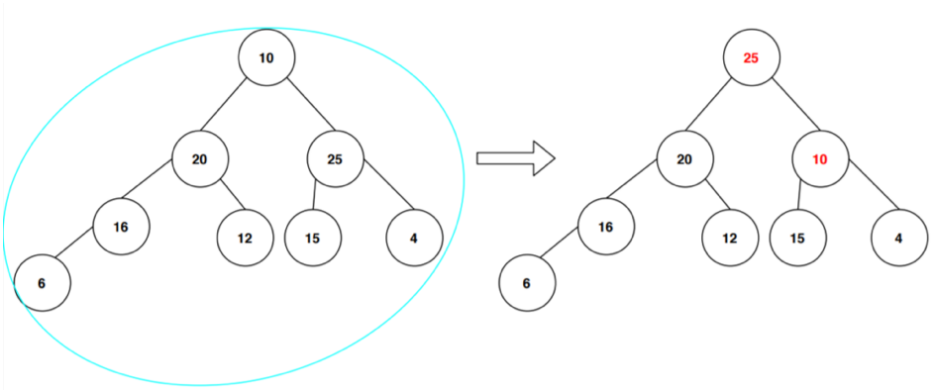
First, we start with the subtree on the lowest level and check if it follows the rules of max heap. Since 16 is greater than 6, we swap the two nodes.



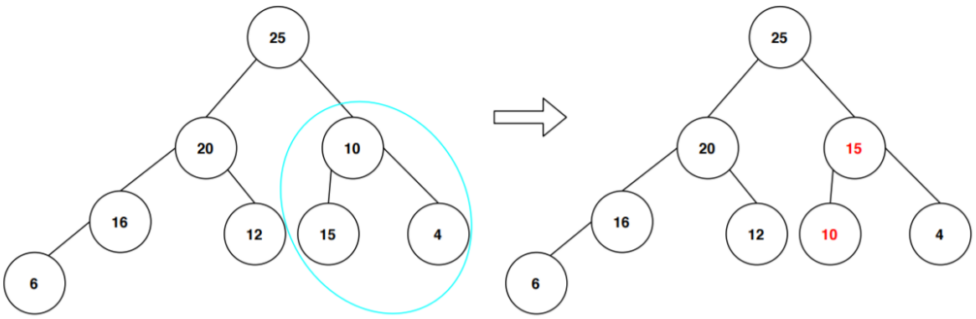
Then, we continue to examine all the subtrees from the lowest level to the top level and it's clear to see the circled subtree is a max heap, thus we don't need to swap anything.



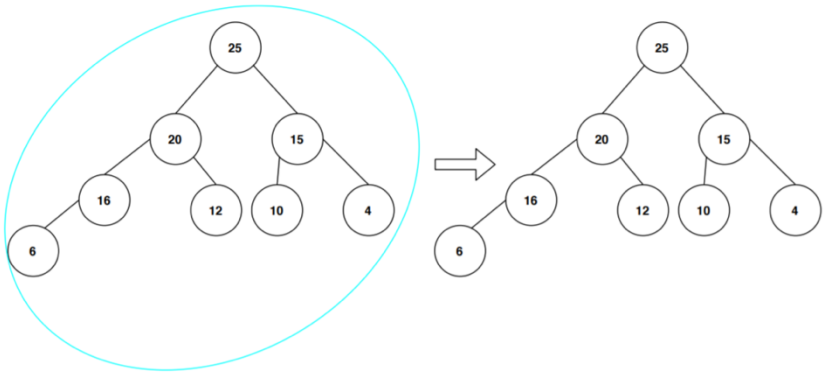
The circled subtree is also already a max heap, so we just leave how it is.



However, when we look at the heap as a whole, the root node contains the value of 10, but both of its children are greater than 10, so we swap 10 and 25 to make it follow the rule of max heap.



Then, we continue to examine the last subtree that is circled, and we swap 10 and 15 to make it obey the rule of max heap.



Finally, we examine the heap as a whole again, and make sure it satisfies the max heap's property. Since it already is a max heap, we just got our final result.

Attached is a pseudocode for max-heapify:

Algorithm 1: Max-Heapify Pseudocode

```
Data: B: input array; s: an index of the node
Result: Heap tree that obeys max-heap property
Procedure Max-Heapify(B, s)
    left = 2s;
    right = 2s + 1;
    if left ≤ B.length and B[left] > B[s] then
        largest =left;
    else
        largest =s;
    end
    if right ≤ B.length and B[right] > B[largest] then
        largest =right;
    end
    if largest ≠ s then
        swap(B[s], B[largest]);
        Max-Heapify(B, largest);
    end
end
```

To max-heapify an array, we first take in the array B and an index of the node s. We start with the node that is at the lowest level and has children. The variable *left* denotes the left child of a node and for the same reason, the variable *right* denotes the right child of a node.

Like the steps we illustrated above, the algorithm keeps checking if all the subtrees obey the rules of max heap until the tree itself is a max heap. In our later analysis part, we will use a real-life example of an emergency room to further integrate the code with practical problems.

The second type of heap sort is called min-heapify. It works the same way as max-heapify, but its ultimate goal is to turn a binary tree into a min heap. Since it has little connection with heap sort, we have attached sources in bibliography, and you are welcomed to read the articles about min-heapify. However, in this report, we will not elaborate on this method.

III. What is heap sort?

After we explained what max-heapify is, it is much easier to understand what heapsort is. To summarize what a heap sort is, it consists of two steps:

1. Build a max heap from a given array.
2. Run max heapify until the tree is a max heap.

The basic idea of a heap sort is first to build a heap and get rid of the root, which is the max value in the heap, by swapping it with the last element of the array. The process is repeated until each element is placed in the right position.

```
HEAPSORT(A)
1   BUILD-MAX-HEAP (A)                          O(n)
2   for i = A.length downto 2                   O(n)
3       exchange A [1] with A [i]               O(1)
4       A.heap-size = A.heap-size – 1           O(1)
5       MAX-HEAPIFY (A, 1)                      O(log n)
```

As we can see from the pseudocode of a heapsort, the first step is to build a max heap, then swap the first element, which happens to be the greatest value in the array, and the last elements. After the swap, the size of elements we need to handle is reduced by 1. Since the max value is taken out, we need to perform a max heapify again to find the largest value in the rest of elements. We repeat the steps again and again, until the number of elements we need to handle is 2.

In our later analysis part, we will continue to talk about the time complexity of building a max heap and max-heapify, further comparing it with other algorithms. We will also talk about advantages and disadvantages of heap sort.

IV. Why does heap sort matter to us?

The reason why we thought of the topic of heap sort is that in the recent CS 5001 and CS 5002 class where we learned about trees. Tree is a fundamental data structure in computer science, and how they can be used to organize and store data efficiently really fascinates us. During our class, the instructor briefly mentioned heap sort as an algorithm that uses a special kind of tree to sort data.

Heap sort is a sorting algorithm that can be used to efficiently sort large databases, create priority queues, optimize network traffic, and perform other computational tasks in computer science. It is especially well-suited to large datasets, with an average and worst-case time complexity of O(nlogn). Additionally, because it is an in-place sorting algorithm, it can perform its operations without requiring extra memory resources.

By selecting heap sort as our topic of our final project, we not only can deepen our understanding of how algorithms and data structures work, but also can further apply these concepts in real-world scenarios. The most important benefit to study heap sort in advance is that we can develop the skills needed to solve complex problems in future fundamental courses like CS 5008 and CS 5800.

Sylvia Deng:
I have had several experiences involving waiting in long queues. I remember waiting for six hours in the emergency room for a stitch operation six years ago and seeing people who arrived later than me but were treated earlier. It was frustrating at that time. I also recall times when I arrived early and checked in online but still had to wait in long lines

to get through the airport security. As I began exploring computer science recently and learned about the concept of sorting in class, I realized that this concept based on priority plays a crucial role in many areas of our lives and finally understood why I need to wait for such a long time.

I have learned about the queue that is first in first out or first in last out in class. Unlike these queues, in priority queue, high priority goes first. I would like to dive into heap sort algorithms, motivated by my experiences. I learned that heap sort is a sorting algorithm that uses a priority queue to sort elements based on their priority. Just like in the emergency room, patients with more serious symptoms are treated before patients with lighter symptoms. I would like to explore more about heap sort and its implementation in computer science and applications in real life. Learning heap sort will give me a deeper understanding of how the emergency center sorts the patients' priorities and how priorities can be used to optimize performance. I believe understanding how heap sort can efficiently sort through large amounts of data will be very useful in a world where data is becoming increasingly abundant and complex.

Wenqian Xie:
Since I attended high school in California, USA, I started my journey of taking airplanes at least twice a year, traveling between China and the United States. During my wait time at the airport, I always notice the interesting fact that some people, even if they do not belong to business or first class, have the priority to get aboard and off than other people in economy class. I was wondering why this happened and I spent the past 10 years watching every passenger who had that priority.

After my observation, I found that people with disabilities or special needs, for example, the elderly, the pregnant, people with wheelchairs or people who need to take care of children, will always get aboard first. In other cases, people who have membership of that airline or have accumulated a lot of flying points can also have the priority. Finally, some military soldiers, probably not just getting aboard and getting off airplanes, they have queue priorities everywhere.

As a result, I realize the airline companies must have its own algorithm and sorting method to assign people with different weights (or importance). It reminds me of the heap, which is used to store and process large datasets, such as thousands of people who're taking the same international airline. Since I am studying computer science and CS 5002 gives me a chance to dive into the heap sort algorithm, I definitely want to take this advantage and figure out how I can, in the future, get the queue priority? So I can spend less time waiting.

Louise Huang:
I used to manually calculate and update the list of hot products to be placed at the top rank of the website when I worked as a marketing manager. Creating a priority queue using heap sort can automate this process and provide real-time data on the most popular products.

With a priority queue implementation, I can assign priorities to each product based on their monthly sales data or other relevant metrics such as user ratings, reviews, or social media buzz. The products with the highest priorities will then be placed at the top of the website and given the most visibility to potential customers. This can help to improve the overall customer experience.

By studying heap sort, I can gain a better understanding of how to optimize the performance of the website and other marketing initiatives. I can use heap sort to identify popular products or customer segments based on relevant metrics such as purchase history or website traffic data. This can help me to tailor the marketing efforts to specific audiences and promote the products that are most likely to generate sales.

Shirley Qiu:
I used to be so fascinated by the "you may also like" feature of shopping applications. Every time I saw those recommended products, I could not help but have a look at them, as the recommended products were often cheaper, more eye-catching, or more to my taste than what I found, which happened a little more frequently than I felt comfortable. It seems “you may also like” knows me better than I do.

This feature provides me with a variety of new products that I never thought about before, which allows me to explore new kinds and types of products that I have never tried. Bad news is that it also helps take a lot of money away from my wallet.

Though the algorithm to its success behind the scenes may vary from app to app, and may not be directly related to Heap Sort, I feel like this feature could take advantage of Heap’s ability to sort great amounts of data and to find the products that best match to customers’ preference based on their previous shopping behaviour or habits.

This personal connection with Heap motivates me to learn more about Heap Sort, and through this final project, I may be able to find out what is responsible for the money I spent (not me obviously).

Data Analysis

Use heap sort to implement priority queues:

We are using the emergency room priority situation to demonstrate how heap sort is an efficient sorting algorithm that can be used to sort a priority queue.

Information of the ten patients:

'name': 'John', 'age': 25, 'priority':, 'symptoms': 'Fever, cough, body aches'
'name': 'Mary', 'age': 35, 'priority':, 'symptoms': 'Difficulty breathing, chest pain'
'name': 'Bob', 'age': 45, 'priority':, 'symptoms': 'Nausea, vomiting, diarrhea'
'name': 'Alice', 'age': 50, 'priority':, 'symptoms': 'Headache, dizziness, fatigue'

'name': 'Chris', 'age': 60, 'priority':, 'symptoms': 'Broken arm, bleeding'
'name': 'Emily', 'age': 30, 'priority':, 'symptoms': 'Severe abdominal pain, fever'
'name': 'David', 'age': 55, 'priority':, 'symptoms': 'Chest tightness, shortness of breath'
'name': 'Samantha', 'age': 65, 'priority': , 'symptoms': 'Unconscious, Unresponsive'
'name': 'Oliver', 'age': 40, 'priority':, 'symptoms': 'Seizures, confusion, disorientation'
'name': 'Julia', 'age': 45, 'priority':, 'symptoms': 'High blood pressure, headache, blurred vision'

To calculate the priority scores based on age and symptoms, we used a weighted formula that considers the severity of the symptoms and the age of the patient. The higher score indicates a higher priority or greater urgency.

Priority score = 0.4 * (100 - age) + 0.6 * (severity of symptoms)

Warning:
The priority score calculation model used in this program is a simplified and potentially biased model. It is not a substitute for professional medical evaluation or diagnosis.

The symptoms component is calculated based on the severity of the patient's symptoms. The severity of the symptoms can be assigned a numerical value based on the table below, with more severe symptoms receiving a higher value.

Table 1:

Score 5 Resuscitation	Conditions that are a threat to life requiring immediate aggressive interventions to restore or preserve life.	· Unresponsive · Near respiratory arrest · Unconscious · Major trauma · Shock
Score 4 Emergent	Conditions that are a potential threat to life, limb, or function requiring immediate intervention.	· Severe respiratory distress · Head injury – with altered mental status · Chest pain – high cardiac suspicion or trauma · Severe trauma, moderate/severe dyspnea

Score 3 Urgent	Conditions that indicate a serious illness or injury requiring intervention. Usually associated with significant distress/discomfort.	· Asthma – respiratory distress · Head injury – no altered mental status · Chest pain – high musculoskeletal suspicion · Seizure alert on arrival
Score 2 Less Urgent	Conditions presenting as an illness or injury requiring intervention. Intervention can be delayed without harmful consequence.	· Asthma – mild respiratory distress · Chest pain – no distress – no cardiac history · Constipation – moderate abdominal distress · Minor lacerations · Minor infections
Score 1 Non Urgent	Conditions that indicate a minor illness or injury for which intervention could be delayed or deferred indefinitely.	· Upper respiratory infections · Return visits to ER for re-check etc.

Then, we add up the values for all of the patient's symptoms to get the severity of symptoms score.

To map the person's age onto a score between 0 and 10, we subtract their age from 100, and then divide the result by 10 to get their age score. This means that younger patients will generally have a higher priority score than older patients.
Once we have both components of the priority score, we add them together to get the final priority score for each patient. We can then use these scores to reorder the emergency_queue in order of priority score.

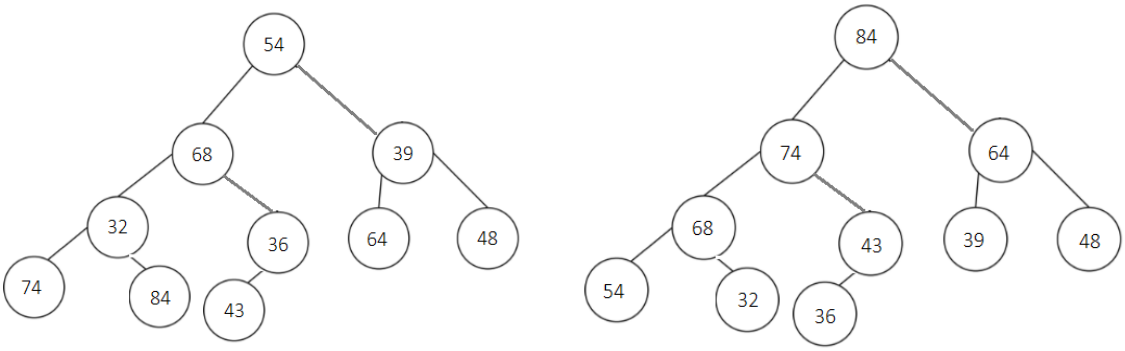
Calculate overall priority scores for each person in the emergency queue:

- John: Age score = (100 - 25) / 10 = 7.5. Fever (2) + Cough(1) +body aches(1)= 4

Priority score = 0.4 * 7.5 + 0.6 * 4 = 5.4

- Mary: Age score = (100 - 35) / 10 = 6.5. Difficulty breathing (3) + Chest pain (4) = 7
  Priority score = 0.4 * 6.5 + 0.6 * 7 = 6.8
- Bob: Age score = (100 - 45) / 10 = 5.5. Nausea (1) + Vomiting (1) + Diarrhea (1) = 3
- Priority score = 0.4 * 5.5 + 0.6 * 3 = 3.9
- Alice: Age score = (100 - 50) / 10 = 5.0. Headache (1) + Dizziness (1) + Fatigue (0) = 2
  Priority score = 0.4 * 5.0 + 0.6 * 2.6 = 3.2
- Chris: Age score = (100 - 60) / 10 = 4.0 Broken arm (2) + Bleeding (2) = 4
  Priority score = 0.4 * 4.0 + 0.6 * 4 = 3.6
- Emily: Age score = (100 - 30) / 10 = 7.0 Severe abdominal pain (4) + Fever (2) = 6
  Priority score = 0.4 * 7.0 + 0.6 *6 = 6.4
- David: Age score = (100 - 55) / 10 = 4.5 Chest tightness (2) + Shortness of breath (3) = 5
  Priority score = 0.4 * 4.5 + 0.6 * 5 = 4.8
- Samantha: Age score = (100 - 65) / 10 = 3.5 Unconscious (5) + Unresponsive(5) =10
  Priority score = 0.4 * 3.5 + 0.6 * 10= 7.4
- Oliver: Age score = (100 - 40) / 10 = 6.0 Seizures (4) + Confusion (3) + Disorientation (3) = 10
  Priority score = 0.4 * 6.0 + 0.6 * 10 = 8.4
- Julia: Age score = (100 - 45) / 10 = 5.5 High blood pressure (2) + Headache (1) + Blurred vision (1) = 4
  Priority score = 0.4 * 5.5 + 0.6 * 4 = 4.3

Next, we want to create a max heap using the patients priority scores



[5.4,6.8,3.9,3.2,3.6,6.4,4.8,7.4,8.4,4.3]→ After Max Heap → [8.4, 7.4, 6.4, 6.8, 4.3, 3.9, 4.8, 5.4, 3.2, 3.6]

Finally, we are going to use heap sort to get the sorted result:
Swap first element with the last element: [3.6, 7.4, 6.4, 6.8, 4.3, 3.9, 4.8, 5.4, 3.2, 8.4]
Continue to max-heapify the rest of array: [3.6, 7.4, 6.4, 6.8, 4.3, 3.9, 4.8, 5.4, 3.2, 8.4]

…… Repeat the steps ……

Until we sort every element and reach: [3.2, 3.6, 3.9, 4.3, 4.8, 5.4, 6.4, 6.8, 7.4, 8.4]

The result of the sorted queue, showing the names of the patients and their corresponding priority scores in descending order.

```
Alice - priority: 3.2
Chris - priority: 3.6
Bob - priority: 3.9
Julia - priority: 4.3
David - priority: 4.8
John - priority: 5.4
Emily - priority: 6.4
Mary - priority: 6.8
Samantha - priority: 7.4
Oliver - priority: 8.4
```

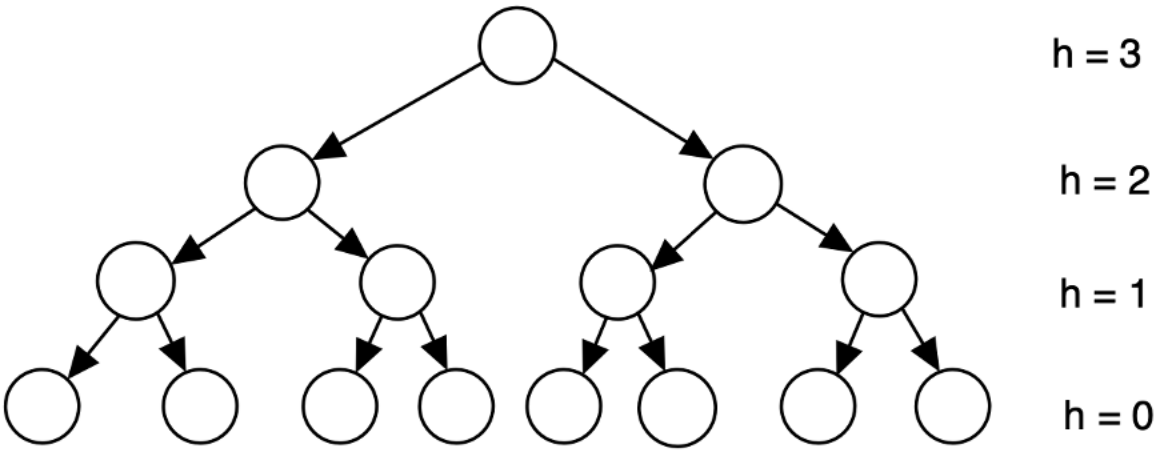
But we want the patients with higher priority are treated first, so we printed the the sorted queue to loop in reverse order:

```
Oliver - priority: 8.4
Samantha - priority: 7.4
Mary - priority: 6.8
Emily - priority: 6.4
John - priority: 5.4
David - priority: 4.8
Julia - priority: 4.3
Bob - priority: 3.9
Chris - priority: 3.6
Alice - priority: 3.2
```

Time Complexity of Heapsort

Heapsort has two phases: Phase 1 is Build Max Heap and Phase 2 is Max Heapify. In this section, we will discuss both time complexity of Phase 1, Phase 2, and the total time complexity of heapsort.

To understand the time complexity of heap sort, we first need to know the height of a heap. The height is important because it determines the running time. The height is defined as the number of edges on the longest downward path from a node to a leaf node. So, the height of a heap is the height of its root. For example, the heap below has a height of three, from the root node to the leaf node. The leaf node level is at the height 0, and then the height increases by one level as we move up towards the root node.



In a heap of height 3 above, we can see that there are 8 leaf nodes, the sum of the nodes of height 1, height 2, and height 3 is 7. It is nearly half of it. So, when we are in heapsort phase 1, the Build Max Heap, we usually start from the last node of the first half, which is the right-most node at height 1. In the worst-case scenario, we swap the element once with its subtree if the node in height 0 is greater. At height 2, we could swap twice. And at height 3, we could swap three times, etc.

Let's assume the total number nodes is equal to $n = 2^k - 1$.

If k= 4, like the graph above, then . The total number of swaps is at most
$8 * 0 + 4 * 1 + 2 * 2 + 1 * 3 = 11$, which is 4 less than 15.

If k = 5, then . The total number of swaps is at most
$16 * 0 + 8 * 1 + 4 * 2 + 2 * 3 + 1 * 4 = 26$, which is 5 less than 31.

Let's take k = 4 as an example, the maximum number of swap operations S is therefore:

$$S(max) = 8 * 0 + 4 * 1 + 2 * 2 + 1 * 3$$

If we multiply both sides by 2, we get:
$$2 * S(max) = 16 * 0 + 8 * 1 + 4 * 2 + 2 * 3$$

Let us place both terms like this:
$$S(max) = 8 * 0 + 4 * 1 + 2 * 2 + 1 * 3$$
$$2 * S(max) = 16 * 0 + 8 * 1 + 4 * 2 + 2 * 3$$

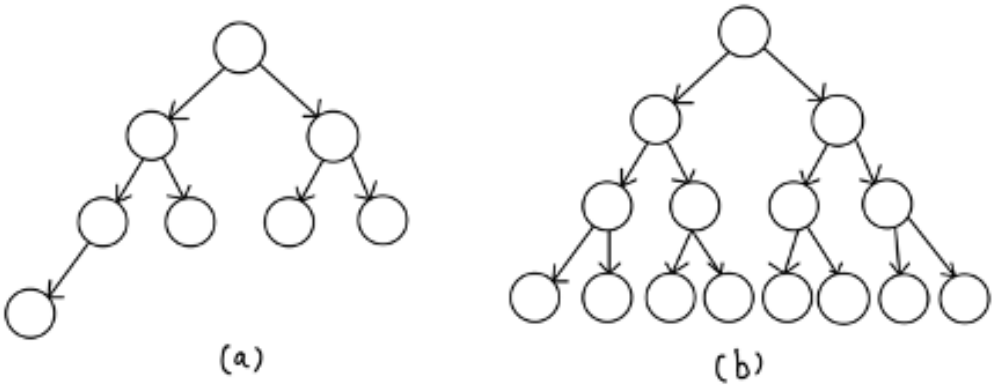
Then we subtract the terms, we get:
$$2 * S(max) - S(max) =$$
$$16 * 0 + 8 * (1 - 0) + 4 * (2 - 1) + 2 * (3 - 2) - 1 * 3$$

Which simplifies to:
$S(max) = (8 + 4 + 2) - 3 = (2^4 - 1) - 4 = 11$

So, if $n = 2^k - 1$, the total number of swaps is:
$S(max) = (2^k - 1) - k$

Therefore, $S(max) < n$, where n is the total number of nodes in the heap. Thus the Build Max Heap runs O(n) time.

Now let's move onto phase 2 of heapsort, Max Heapify.



For heap (a) and (b) above, we can see that heap (a) has a minimum number of nodes in the heap of height 3 and heap (b) has a maximum number of nodes in the heap of height 3. When height = 3, the minimum of nodes is $2^3 = 8$ and the maximum of nodes is $2^4 - 1 = 2^{3+1} - 1 = 15$.

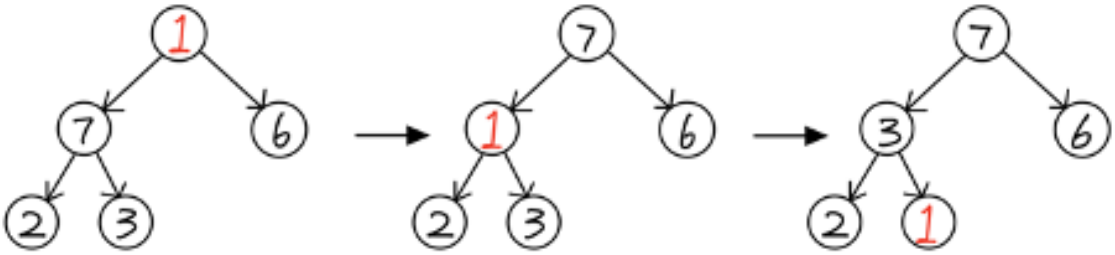
Therefore, the minimum of nodes in the heap is 2^h and the maximum of nodes in the heap is $2^{h+1} - 1$ in the heap of height h.

This means that the number of nodes is in between the minimum number and the maximum number in the heap of height h. So, the range is:
$2^h <= n <= 2^{h+1} - 1$

Then we can simplify this by removing the and the equal sign. In other words, we added 1 on the right side so it must be greater than n. So, it can be simplified to:
$2^h <= n < 2^{h+1}$

And now it is also equal to:
$h <= log\, n < h + 1, where\, h = height$

We can see that the height is equal to (log n). When the heap is an invalid max heap, Max Heapify will be called to maintain the max heap property. What it does is to let the value which is smaller bubble down until the subtree obeys the max heap property.



From the figure above, element 1 is originally at the root node. It bubbles down from the root to the leaf node. The figure above actually showed the worst-case scenario, moving the root node all the way to the leaf node level. The time complexity of calling Max-Heapify is related to the height of the heap. This implies Max-Heapify is O(log n), where n is the total number of elements in the entire tree. Recall how heapsort works from the previous introduction section, the Max-Heapify will be called (n-1) times, each of which takes O(log n) time. Therefore the time complexity of Max-Heapify is O(n log n).

As a result, heapsort is consisting of two steps:

1. Build Max Heap to turn the input into a heap, which takes O(n) time.
2. Max-Heapify to maintain the sorted set, which takes O(n log n) time.

So, the total time complexity of heapsort is O(n) + O(n log n) and we can drop O(n) as it is a low-order term. That is, the total time complexity of heapsort is O(n log n), for any and every input set with n elements.

Since it always follows the same steps irrespective of the input array order, the time complexity remains O(n log n) for the best case, average case, and worst case.

Case	When it occurs	Time Complexity
Best case	When there is no sorting required	O(n log n)
Average case	When elements are neither fully ascending nor descending	O(n log n)
Worst Case	When elements are required to be sorted in reverse order	O(n log n)

Heap Sort Algorithm(n.d.)

Comparison

Heap Sort vs. Bubble Sort

	Heap Sort	Bubble Sort
How it works	Convert them into a heap and swap the last one with the root. It repeats the same process.	Iterate through the array repeatedly and compare adjacent elements and swap them if needed until the array is sorted
Time complexities	Best case, Average case and Worst case: O(n log n)	Best case: O(n) Average case: O(n^2) Worst case: O(n^2)

Bubble Sort vs Heap Sort(2022)

Bubble sort can be efficient and effective when there are a small number of elements in the array, and the array is mostly sorted. This is because the swapping times will be less (Soni Upadhyay, 2023). In addition, with the best case time complexity, bubble sort can be useful for determining whether a list is already sorted or not (Soni Upadhyay, 2023). However, it becomes highly inefficient for large data sets, as we can see from the table above, where the worse-case time complexity is O(n^2). Therefore, heapsort is a much better choice for sorting large datasets because even in the worst case scenario, the time complexity is still O(n log n).

Advantages and Disadvantages of heap sort

Heap sort is an efficient sorting algorithm, as we just proved that its time complexity is always O(nlogn), in both the best and worst scenarios. As a result, with the increase of the size of the data, the time needed grows proportionally, which makes it better at sorting for large datasets, while the time complexity of other algorithms, like quick sort, will be O(n^2) when facing the worst case.

Memory usage is minimal for heap sort because it doesn't need extra memory space when the algorithm is executed. As we just mentioned, heap sort is used for sorting large datasets, it is good to know we don't need to worry about running out of memory when sorting.

However, heap sort is costly for small data sizes, as we have to construct a max heap first, which requires additional steps before actually doing the sorting. Another disadvantage of heap sort is that it does not care about the original order of the list. As a result, if the list is fully or nearly sorted, heap sort stills creates a max-heap first and then heapify it, which can be inefficient, so we'd better use a different sorting algorithm in such cases.

Conclusion

Weaknesses

One weakness is that our project mainly focuses on the technical part of heap sort. Though we did some research about a real-world application, the emergency room case, we did not consider other usages of heap sort.

Besides, we only provided a small data set of 10 elements for our project, but in reality, the datasets are usually much larger, as heap sort is known as costly and inefficient when sorting data of small sizes.

These weaknesses suggest that we can do more research using larger datasets and explore other potential scenarios beyond the emergency room case in the future to better understand this sorting algorithm.

Limitation

One of the limitations of our project is that it provides limited insights, as we only compared the time complexity of heapsort and bubble sort algorithms. This may not provide a deep understanding of the strengths and weaknesses of each algorithm. There are still more aspects that are not captured in the comparison such as space complexity and stability. Future research could explore the performance of other algorithms, such as quicksort, merge sort, or insertion sort etc in terms of their time complexity, performance, memory usage, stability, adability and other factors.

What we learned?

Sylvia:
Completing this project on heapsort has been a valuable experience for me. I gained a deeper understanding of heapsort and its performance. Through researching this topic and watching educational videos, I learned the in-depth analysis of the time complexity of heapsort. This deeper understanding of heapsort will undoubtedly be useful as I progress through my studies, particularly for future courses like CS 5008 and CS 5800, where data manipulation and optimization play a vital role. It is fascinating that an algorithm like heapsort could be used to address the complex problem of prioritizing patients in emergency rooms. With numerous people requiring urgent medical treatment today, it is crucial to have an efficient system in place to ensure patients get treated in a timely manner.

Louise:
Throughout the project, I gained a much deeper understanding of how heap sort works, how its importance in sorting and prioritizing data efficiently, and how to implement heap sort algorithm using Python.This project has shown me the power of computational thinking and the importance of optimization, which will undoubtedly be useful as I progress through my studies and in my future career.

Wenqian:
By completing the research of heap sort, I improved my problem-solving skills by separating a big problem into small steps. In the beginning, I didn’t even know what a heap was. From heap, to heapify and finally heapsort, I gained a better understanding of the fundamental rules of heapsort. Heapsort can look intimidating at first looking at so many trees, however, it only consists of two steps but conducted recursively. In my future learning, whether it’s courses or professional career, even if I encounter something I don’t understand at first, I won't panic because I know no matter how difficult it looks, it will be familiar once it breaks down.

Shirley:
In our project, we analyzed the priority queue for emergency rooms. Despite the small size of the dataset we used, the same algorithm can be used with datasets that are much larger. Our project has helped me feel more prepared for the upcoming CS 5008, as through the process, I have gained a deeper understanding of heap sort and other sorting algorithms. Another important lesson I learned from the project is the problem-solving skill. I feel natural and confident starting with smaller data sizes to better understand the problem at hand. This approach helps me discover efficient algorithms that may not be so obvious when working with larger datasets.

References

1. Bubble sort vs heap sort. Bubble Sort vs Heap Sort - TAE. (n.d.). Retrieved April 4, 2023, from https://www.tutorialandexample.com/bubble-sort-vs-heap-sort
2. Heap sort - javatpoint. www.javatpoint.com. (n.d.). Retrieved April 10, 2023, from https://www.javatpoint.com/heap-sort
3. Upadhyay, S. (2023, February 20). What is bubble sort algorithm? Time Complexity & Pseudocode: Simplilearn. Simplilearn.com. Retrieved April 12, 2023, from https://www.simplilearn.com/tutorials/data-structure-tutorial/bubble-sort-algorithm
4. Datta, W. by: S. (2022, November 11). Max-heapify a binary tree. Baeldung on Computer Science. Retrieved April 2, 2023, from https://www.baeldung.com/cs/binary-tree-max-heapify
5. GeeksforGeeks. (2023, April 24). Heap sort. GeeksforGeeks. Retrieved April 6, 2023, from https://www.geeksforgeeks.org/heap-sort/
6. GeeksforGeeks. (2023, April 17). Binary heap. GeeksforGeeks. Retrieved April 16, 2023, from https://www.geeksforgeeks.org/binary-heap/
7. How emergency cases are prioritized: Kemptville District Hospital. Kemptville District Hospital | Building Healthier Communities. (2016, November 13). Retrieved April 15, 2023, from https://www.kdh.on.ca/our-services/24-hour-emergency/how-emergency-cases-are-prioritized/
8. Wikimedia Foundation. (2022, November 24). *Heapsort*. Wikipedia. Retrieved April 20, 2023, from https://en.wikipedia.org/wiki/Heapsort
9. Wikimedia Foundation. (2022, November 24). *Heapsort*. Wikipedia. Retrieved April 2, 2023, from https://en.wikipedia.org/wiki/Heapsort
10. GeeksforGeeks. (2023, March 6). *Applications, advantages and disadvantages of Heap*. GeeksforGeeks. Retrieved April 13, 2023, from https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-heap/