Heapsort

Hausarbeit für

TI2AD: Algorithmen und Datenstrukturen

in der Studienrichtung Informatik

Inhaltsangabe

1 Einführung

Der Heapsort zählt zu den Sortieralgorithmen. Ein solcher Algorithmus besteht aus einer Reihe von Anweisungen, welche die Elemente eines Arrays in eine bestimmte Reihenfolge bringen. Der hier betrachtete Algorithmus zählt zu den vergleichsbasierten Sortieralgorithmen, was bedeutet, dass nach jedem Schritt des Sortierens entschieden wird, ob ein Element rechts oder links eines anderen Elements stehen soll. Für dieses Sortieren verwendet der Heapsort die Datenstruktur „Heap“ auf welche nachfolgend eingegangen wird. [1]

2 Voraussetzungen zur Anwendung des Heapsort

Zur Verwendung des Heapsort muss das zu sortierende Arrays zunächst in einen Max Heap umgewandelt werden. [2]

> Ein "Heap" (deutsch: "Haufen" oder "Halde") bezeichnet einen Binärbaum, in dem jeder Knoten entweder größer/gleich seiner Kinder ist ("Max Heap") – oder kleiner/gleich seiner Kinder ("Min Heap"). [2]

Zuerst wird das gegebene Array gedanklich auf einen Binärbaum übertragen. Dies dient jedoch nur der Darstellung.

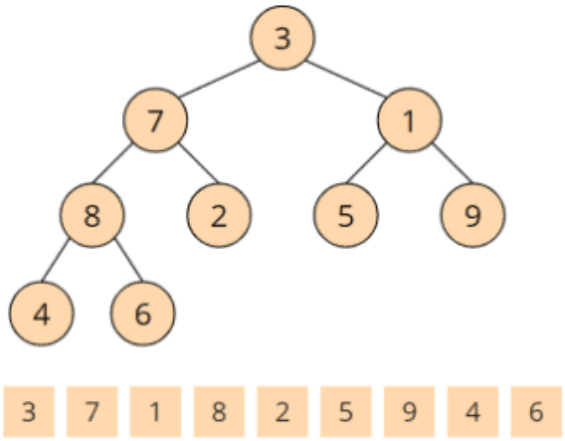

Abbildung 1: Projektion eines Arrays auf einen Binärbaum [2]

Nun muss dieser Binärbaum so umsortiert werden, dass die Elternknoten immer größer oder gleich ihrer Kinder sind. Dadurch erhält man einen Max Heap. Dafür werden rückwärts alle Elternknoten mit einer rekursiven heapify-Funktion abgearbeitet.

Diese Funktion prüft für jeden Elternknoten die Heap-Bedingung und verändert gegebenenfalls das Array. Auf diesen Prozess wird am folgenden Beispiel der Zahlenfolge [3, 7, 1, 8, 2, 5, 9, 4, 6] weiter eingegangen. Zur besseren Lesbarkeit wird auf die Verwendung von Zahlwörtern verzichtet.

Der letzte Elternknoten ist die 8. Die heapify-Funktion vergleicht den linken und rechten Kindknoten, welche sich im Array an den Stellen $n * 2 + 1$ und $n * 2 + 2$ befinden, jeweils mit dem Elternknoten

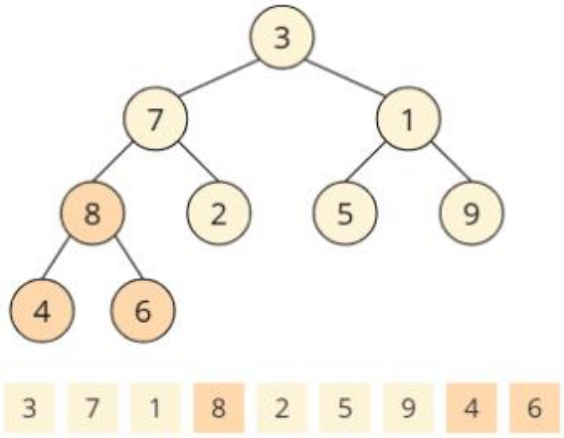

Abbildung 2: Aufruf der heapify-Funktion auf letzten Elternknoten [2]

In diesem Fall ist die Heap-Bedingung bereits erfüllt, da der linke Kindknoten 4 und der rechte Kindknoten 6 bereits kleiner als der Elternknoten 8 sind. Damit ist der erste Knoten abgearbeitet.

Der nächste zu betrachtende Elternknoten ist der vorletzte. Dieser ist in diesem Fall die 1.

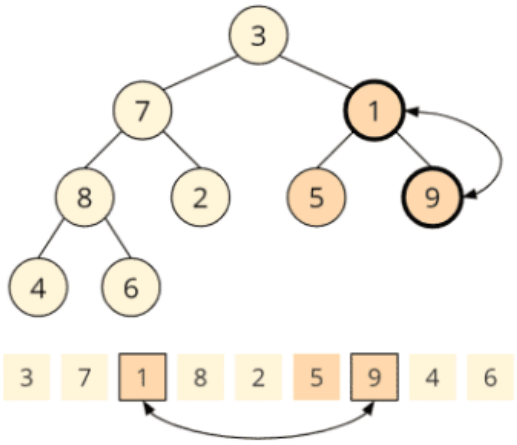

Abbildung 3: Aufruf auf vorletzten Elternknoten [2]

Die Kindknoten von 1 sind 5 und 9. Das bedeutet die Heap-Bedingung ist verletzt und der Elternknoten muss getauscht werden. Da beide Kindknoten größer sind, müssen diese vorher noch miteinander verglichen, sodass der größte Kindknoten der neue Elternknoten wird. Der neue Elternknoten ist nun die 9. Hiermit ist auch dieser Knoten abgearbeitet.

Darauf folgt die Bearbeitung des Elternknoten 7.

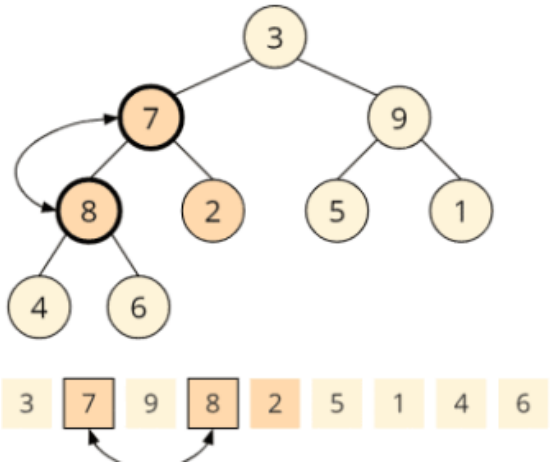

Abbildung 4: Aufruf auf Elternknoten 7 [2]

Die Kindknoten sind 8 und 2. Die 2 ist kleiner als die 7, was die Heap-Bedingung erfüllt. Jedoch ist die 8 größer als die 7. Deswegen wird die 8 zum neuen Elternknoten und auch hier ist die Heap-Bedingung damit erfüllt. Da jedoch die 7 nun selbst wieder weitere Kindknoten hat, muss auch für diese noch einmal die Bedingung geprüft werden. Die 7 ist größer als die 4 und die 6, womit dieser Knoten beendet ist.

Zuletzt wird der erste Knoten, die 3, betrachtet.

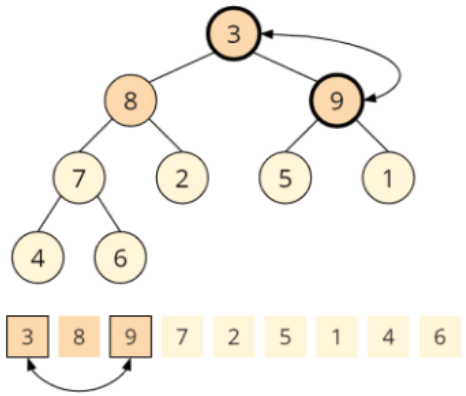

Abbildung 5: Aufruf auf ersten Knoten [2]

Hier sind wieder beide Kindknoten größer als der Elternknoten, was bedeutet, dass der größte Kindknoten, die 9, mit dem Elternknoten 3 getauscht wird. Auch hier muss wieder überprüft werden, ob die neuen Kindknoten der 3 die Heap-Bedingung verletzen.

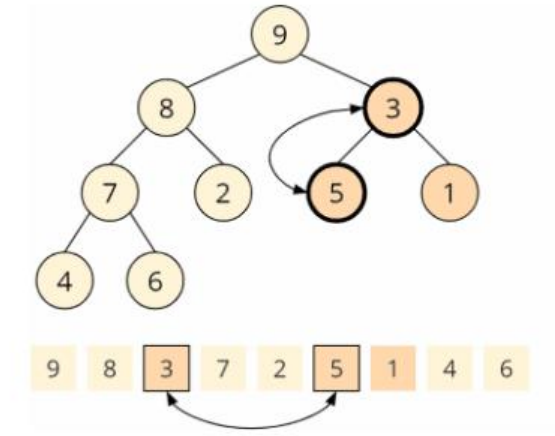

Abbildung 6: Aufruf auf getauschten Knoten [2]

Die 5 und die 3 müssen noch getauscht werden und die heapify-Funktion ist damit beendet. Der Binärbaum wurde in einen Max Heap umgewandelt und die Voraussetzung zur Ausführung des Heapsort-Algorithmus ist gegeben.

3 Beschreibung des Algorithmus

Der Algorithmus benutzt nun in der sort-Funktion den vorher erstellten Max Heap und tauscht im ersten Schritt den Wurzelknoten, welcher an erster Stelle im Array steht, mit dem letzten Knoten am Ende des Arrays. [2]

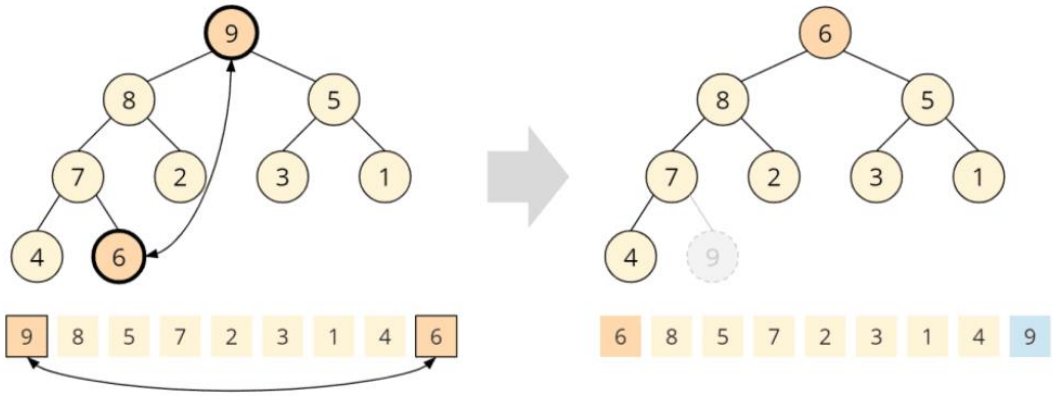

Abbildung 7: Tauschen von Wurzelknoten mit letztem Knoten [2]

In unserem Beispiel ist dies, wie in Abbildung 7 zu sehen, die 9 und die 6. Weiterhin wird die 9 gedanklich aus dem Heap entfernt.

Da nun jedoch die Heap-Bedingung nicht länger erfüllt ist, muss im nächsten Schritt mit der heapify-Funktion diese wieder hergestellt werden.

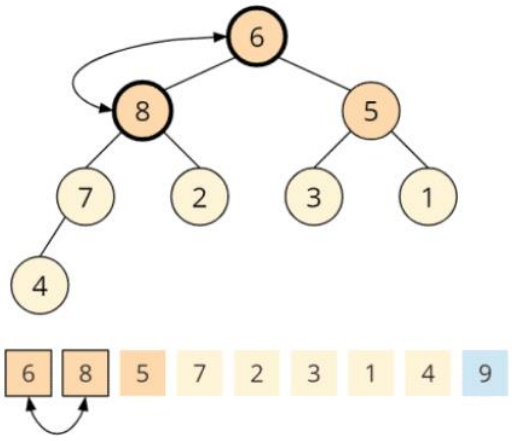

Abbildung 8: Wiederherstellung der Heap-Bedingung [2]

Das bedeutet die eben als Wurzelknoten eingefügte 6 muss mit ihrem Kindknoten 8 getauscht werden. Da die 6 an ihrer neuen Position jedoch wieder Kindknoten besitzt, muss auch hier die Heap-Bedingung geprüft werden.

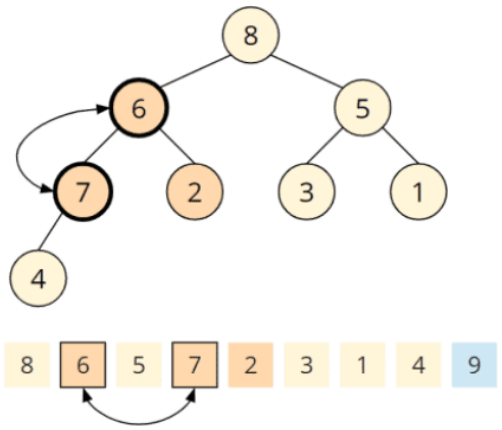

Abbildung 9: Tausch von 6 und 7 [2]

Die 7 ist größer als die 6 und damit muss erneut getauscht werden. Abschließend wird die 6 noch mit der 4 verglichen. Hier stimmt jedoch die Position der Knoten bereits und ein Max Heap ist wieder hergestellt.

Anschließend wird der ganze Prozess für den neuen Wurzelknoten ausgeführt.

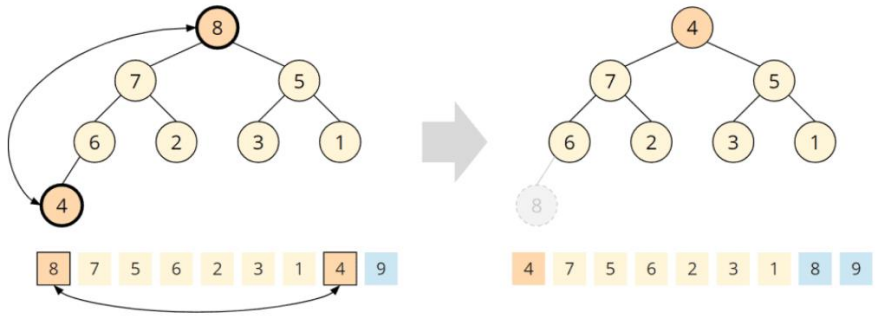

Abbildung 10: erneutes Tauschen des Wurzelknotens mit letzten Knoten [2]

Da der Baum bereits um ein Element gekürzt ist, ist der letzte Knoten an vorletzter Stelle des Arrays. Dieser muss nun mit dem Wurzelknoten getauscht werden.

Die 8 wird damit auch aus dem Baum entfernt und nachfolgend muss erneut über die heapify-Funktion die Heap-Bedingung hergestellt werden. Nach diesem Schritt sind die letzten 2 Elemente des Arrays sortiert.

Diese Schritte werden nun so lang wiederholt, bis der Baum nur noch ein Element enthält

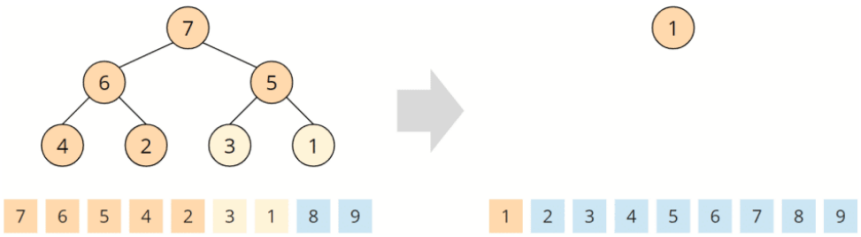

Abbildung 11: Wiederholung der Schritte [2]

Ist dies erledigt, so ergibt sich ein sortiertes Array und der Algorithmus ist damit beendet.

4 Eigenschaften des Heapsort

Im Nachfolgenden wird auf die Eigenschaften Zeitkomplexität, Platzkomplexität, Stabilität und Parallelisierbarkeit eingegangen. [2]

Die Zeitkomplexität wird in der O-Notation angegeben. Sie setzt sich beim Heapsort aus der Zeitkomplexität der heapify-Funktion, derer Aufrufe und der buildheap-Funktion zusammen. Die heapify-Funktion arbeitet sich einmal durch den Baum. Da die Höhe eines Binärbaums maximal $\log_2 n$ beträgt die Komplexität dieser Funktion $O(\log n)$. Die heapify-Funktion wird $n-1$ mal aufgerufen, weswegen die Komplexität für die Aufrufe dieser Funktion $O(n \log n)$ beträgt. Die buildheap-Funktion ruft für jeden Elternknoten die heapify-Funktion auf. Ein Heap besitzt $n/2$ Elternknoten. Daraus folgt, dass die Komplexität der heapify-Funktion maximal $O(n \log n)$ ist. Beide Teile haben somit die gleiche Zeitkomplexität, was bedeutet, dass die gesamte Zeitkomplexität des Heapsort auch $O(n \log n)$ beträgt.

Die Platzkomplexität ist auch in der O-Nation angegeben.

> „Heapsort ist ein In-Place-Sortierverfahren, d. h. außer für Schleifen- und Hilfsvariablen wird kein zusätzlicher Speicherplatz benötigt.“ [2]

Das bedeutet, dass die Platzkomplexität $O(1)$ beträgt.

Die Stabilität ist gegeben, wenn der Sortieralgorithmus die Reihenfolge gleicher Schlüssel bewahrt. Dies ist beim Heapsort nicht der Fall. Damit ist er instabil. [3]

Ein Algorithmus ist parallelisierbar, wenn mehrere Schritte nebeneinander ausgeführt werden können, ohne Abweichende Resultate hervorzubringen. [4]

> „Bei Heapsort wird das gesamte Array ständig verändert, so dass es keine naheliegenden Lösungen gibt den Algororithmus zu parallelisieren.“ [2]

5 Programmbeschreibung

Zur Umsetzung des Algorithmus in ein lauffähiges Programm, wurde sich aufgrund der simplen Syntax für die Programmiersprache „Python“ entschieden. Dementsprechend wird es über die Konsole mit dem Befehl „python3 Heapsort.py“ gestartet. Beim Start wird die Eingabe einer Zahlenfolge erwartet. Diese soll getrennt durch Leerzeichen eingegeben werden. Darauf folgt eine Entscheidung, ob die gegebene Zahlenfolge als Heap durchsucht werden, oder ob der Heapsort auf sie angewendet werden soll.

Beim Suchen wird zuerst die buildheap-Funktion aufgerufen, um aus der Zahlenfolge einen Max Heap zu erstellen und anschließend wird der zu suchende Schlüssel abgefragt. Nach dieser Eingabe erfolgt der Aufruf der search-Funktion, welche mittels einer rekursiven Hilfsfunktion eine Binärsuche durchführt. Zum Schluss erfolgt die Ausgabe an welcher Stelle der Schlüssel im Heap liegt.

Bei Auswahl des Sortierens wird der Heapsort über sort-Funktion gestartet. Diese startet zu Beginn ebenfalls die buildheap-Funktion, um einen Max Heap zu erhalten. Dann iteriert sie rückwärts über das Heap-Array und tauscht, wie in der Beschreibung des Algorithmus erklärt, die Elemente des Heap und ruft die heapify-Funktion, welche die Heap-Bedingung prüft und wiederherstellt, auf. Abschließend gibt sie die sortierte Zahlenfolge aus.

6 Zusammenfassung

Diese Arbeit war eine Vorstellung des Heapsort-Algorithmus. Nach einer Einführung, welche kurz erklärte was der Heapsort ist, ging man auf die Voraussetzungen für dessen Einsatz ein. Unter diesem Punkt wurde das Konzept Heap erklärt und darauf eingegangen, wie man einen erstellt. Darauf folgte eine Beschreibung der Schritte, die bei einem Heapsort ausgeführt werden, um eine Zahlenfolge zu sortieren. Als nächstes folgte eine Vorstellung der Eigenschaften des Heapsort. Zuletzt beschrieb man die Umsetzung des Algorithmus in Python und wie dieses Programm verwendet wird.

7 Verzeichnisse

7.1 Abbildungsverzeichnis

7.2 Quellenverzeichnis

[1]: Sorting Algorithms. Brilliant.org.
https://brilliant.org/wiki/sorting-algorithms/#sorting-algorithms.
(Zugriff am 24.03.2023)

[2]: Heapsort – Algorithmus, Quellcode, Zeitkomplexität. Happycoders.
https://www.happycoders.eu/de/algorithmen/heapsort/.
(Zugriff am 24.03.2023)

[3]: Stabilität (Sortierverfahren). Wikipedia: Die freie Enzyklopädie.
https://de.wikipedia.org/wiki/Stabilit%C3%A4t_(Sortierverfahren).
(Zugriff am 26.03.2023)

[4]: Nebenläufigkeit. Wikipedia: Die freie Enzyklopädie.
https://de.wikipedia.org/wiki/Nebenl%C3%A4ufigkeit.
(Zugriff am 26.03.2023)

8 Selbstständigkeitserklärung

„Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder veröffentlicht, noch einer anderen Prüfungsbehörde vorgelegt.“