Alp2 - Übungsblatt 4

Bearbeitet von: Jasmine Cavael & Alexander Chmielus
Tutor: Fabian Halama
Tutorium 10 (Do. 16-18)

Aufgabe 2a)

(e = erste, z = zweite, d = dritte, v = vierte, f = fünfte, s = sechste)

Die zu sortierende Liste ist [4e, 4z, 4d, 4v, 4f, 4s], da wir aber die Heapstruktur verwenden, muss L[0] mit 0 initialisiert werden.

H = [0, 4e, 4z, 4d, 4v, 4f, 4s] =

Da alle Elemente gleich sind, wird vor der ersten Schleife in Heapsort(H) nichts verändert, außer dass H[0] = 6 gesetzt wird. Desweiteren ändert die Funktion max_heapify nie etwas am Baum, da ja alle Elemente gleich groß sind, wodurch wir immer einen Max-Heap haben. Jetzt geht es in die Schleife.

i = 6:

H[6], H[1] = H[1], H[6]

H[0] -= 1

--> H = [5, 4s, 4z, 4d, 4v, 5f, 4e]

i = 5:

H[5], H[1], H[1], H[5]

H[0] -= 1

--> H = [4, 4f, 4z, 4d, 4v, 4s, 4e]

i = 4:

H[4], H[1], H[1], H[4]

H[0] -= 1

--> H = [3, 4v, 4z, 4d, 4f, 4s, 4e]

i = 3:

H[3], H[1], H[1], H[3]

H[0] -= 1

--> H = [2, 4d, 4z, 4v, 4f, 4s, 4e]

i = 2:

H[2], H[1], H[1], H[2]

H[0] -= 1

--> H = [1, 4z, 4d, 4v, 4f, 4s, 4e]

i = 1:

Schleife wird abgebrochen

H[0] -= 1

--> H = [0, 4z, 4d, 4v, 4f, 4s, 4e]

Wie man sieht, ist die erste 4 nun hinter allen anderen, wodurch die Nicht-Stabilität gezeigt wurde.

Aufgabe 4)

Das kommt ganz drauf an, was mit "effizient" gemeint ist.
Geht es rein um Schnelligkeit, dann dürfte Radixsort (zusammen mit Countingsort) am besten sein, da es in O(n) liegt. Radixsort darf verwendet werden, da wir hier Integerzahlen haben. Wenn man allerdings an den Speicher denkt, wird das ganze schwieriger. 1 Million 32-bit Integer bräuchten 4 MB RAM. Da Radixsort zusätzlichen Speicher benötigt, wird das ganze also schwer umzusetzen sein.
Gehen wir also den Kompromiss ein, dass es langsamer sein darf, wenn dafür weniger Speicheraufwand besteht. Dann würden wir einen in-place Mergsort Algorithmus verwenden. Dieser würde immer in O(n*log(n)) liegen und keinen zusätzlichen Speicher beanspruchen. Desweiteren müssten wir uns die Eingabe genau angucken. Wenn die Liste bereits fast sortiert ist, könnte Insertsort eine option sein, da es in solchen Fällen in O(n) liegt und ebenfalls keinen weiteren Speicher benötigt. Wenn sehr viele Zahlen wiederholt werden und die größte Zahl sehr klein (bspw. 100, oder 1000) ist, würde Countingsort alleine schon sehr schnell sein, bräuchte aber zusätzlichen Speicher.

Da wir aber keine weiteren Angaben bekommen, gehen wir hier von einem Speicher aus, der groß genug ist, um uns die schnelle Radixsort-Variante zu erlauben.