Verslag Sorteeralgortimen

Heapsort - Schoolproject

(de GitHub repository vind je hier: https://github.com/Bakker-Ei/Sorting-algorithms)

Gemaakt door: Yannis, Selton en Maurits

Inleiding

Voor dit project hebben wij het sorteeralgoritme **Heapsort** onderzocht en zelf gecodeerd in Python. hier hebben we zowel gekeken **hoe** het sorteer algoritme werkt, we hebben gekeken naar de **complexiteit** en ook hoe we het konden **visualizeren**

Tijdens dit project hebben we drie verschillende scripts ontwikkeld:
1. Een interactieve variant van Heap Sort.
2. Een snelle variant om de snelheid te testen
3. Een visuele variant waarin het sorteerproces met **Matplotlib** en **NetworkX** wordt weergegeven.

De drie versies zijn te vinden op GitHub:
- Snelle sort
- Interactive sort
- Visuele sort

Ontwikkelproces

We begonnen met het idee om een eenvoudige Heap Sort te schrijven waarbij je een lijst via input kon invoeren. Deze werd vervolgens gesorteerd in een **min-heap** of **max-heap**. Al snel merkten we dat het handig zou zijn om ook een **snelle versie** te maken die enkel het sorteren uitvoert, zonder visuele of interactieve elementen. Dat bleek nuttig bij het berekenen van de **complexiteit**.

We wilden daarnaast ook graag zien hoe Heap Sort er grafisch uitziet. Selton schreef eerst een script dat bij elke stap de integers printte in de terminal, maar

deze stonden niet netjes gecentreerd als binaire boom en misten verbindingslijnen. Yannis probeerde dit te verbeteren door slashes toe te voegen, maar dat maakte de uitlijning nog ingewikkelder. Toen leek het project even vast te lopen — tot het idee ontstond om het **visueel te maken met Matplotlib en NetworkX**.

Na intensief experimenteren lukte het om met Matplotlib de knopen (nodes) te plaatsen en met NetworkX de verbindingen (edges) te tekenen. Dit proces duurde 7 uur, maar uiteindelijk hadden we een werkende visualisatie

Het logboek van dit hele proces is te lezen in Ons Logboek.

Uitleg van het algoritme

Heap Sort bestaat uit twee hoofdfasen:
1. Het opbouwen van een **heap** (een binaire boom waarin elke ouder groter is dan zijn kinderen bij een max-heap).
2. Het telkens verwijderen van het grootste element (de root) en dit verplaatsen naar het einde van de lijst.

Het proces verloopt als volgt:
- De invoerlijst wordt omgevormd tot een max-heap via de *heapify*-functie.
- Het grootste element wordt steeds verwisseld met het laatste ongesorteerde element.
- De heap wordt verkleind en opnieuw geheapified.
- Dit herhaalt zich tot de hele lijst gesorteerd is.

De pseudocode van ons algoritme is hier te vinden.

Complexiteit

Theoretisch gezien:
- Best case: O(n log n)
- Worst case: O(n log n)

Heap Sort werkt meestal in O(n log n) tijd.

Soms hoeft het algoritme minder stappen te doen omdat de lijst al deels een goede heap is, Dit maakt dat sommige lijsten iets sneller gesoorteerd worden, maar de tijd blijft in grote lijnen O(n log n).

In tegenstelling tot veel andere algortimen heeft Heap Sort dus een **stabiele tijdscomplexiteit**, met kleine verschillen afhankelijk van de beginvolgorde van de lijst.

Snelheidsmetingen

Toen de visualisatie werkte, konden we Heap_sprint.py gebruiken om de **complexiteit in de praktijk** te testen. Met behulp van `time.perf_counter` hebben we de sorteertijd gemeten voor verschillende lijsten, telkens met veel herhalingen. De resultaten werden uitgezet in een grafiek met Matplotlib.

De uiteindelijke grafiek toont een duidelijke **O(n log n)**-lijn, wat overeenkomt met de theoretische verwachting. alle snelheidsmetingen zijn uigevoerd op ***Yannis*** zijn laptop, om bij iedere bugfix vergelijkbare resultaten te krijgen

Onze grafiek: