TUGAS MANDIRI

PERANCANGAN & ANALISIS ALGORITMA

“Heapsort”

202323430048

DOSEN PENGAMPU:

Randi Proska Sandra, S.Pd, M.Sc

OLEH:

Zidan Raihanza Rafi

22343032

Informatika (NK)

PRODI SARJANA INFORMATIKA

DEPARTEMEN TEKNIK ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2024

A. PENJELASAN SINGKAT

Heap Sort adalah algoritma sortir seleksi yang ditingkatkan .Ini dilakukan pada data heap dan heap pada dasarnya adalah pohon biner lengkap. Ada 2 sifat dasar heap yaitu maks dan min . Di maxi heap, node induknya lebih besar daripada anaknya dan di tumpukan mini, simpul induknya lebih sedikit dari anaknya . Gambar 2 menunjukkan proses kerja heap sort. Lakukan juga analisa pada kedua Central Processing Unit (CPU) dan Unit Pemrosesan Grafis GPU. Mereka mengerjakannya penyortiran penyisipan, pengurutan gelembung, pengurutan cepat, dan pengurutan heap dan gabungkan sortir. Mereka menyimpulkan bahwa masukan dalam bilangan bulat membutuhkan waktu lebih sedikit dibandingkan dengan string. Menurut mereka temuan, pengurutan seleksi, dan pengurutan cepat tidak digunakan jenis data yang besar.

Dr. I. Lakshmi, melakukan analisis terhadap empat jenis yang berbeda algoritma seperti Quick sort, insertion sort, heap sort, merge menyortir. Peneliti menganalisis kompleksitas waktu Analisis penelitian ini adalah untuk menentukan algoritma mana yang berguna ketika kita memiliki kumpulan data yang membingungkan. Analisisnya adalah tergantung pada kasus terbaik, rata-rata, dan terburuk. Dalam analisis peneliti menggunakan lingkungan C# dan datanya acak digunakan .

Meskipun algoritma heapsort memiliki kompleksitas waktu O(n log n), di mana n adalah jumlah elemen dalam array, tetapi algoritma ini memiliki keuntungan tertentu dibandingkan dengan algoritma pengurutan lainnya seperti mergesort dan quicksort. Keuntungan utama heapsort terletak pada penggunaan memori yang lebih efisien karena tidak memerlukan alokasi memori tambahan seperti yang dibutuhkan oleh mergesort. Selain itu, heapsort juga memiliki keunggulan dalam kinerja dalam beberapa kasus karena struktur heap memungkinkan akses langsung ke elemen terbesar atau terkecil dalam waktu konstan, tanpa memerlukan pengurutan seluruh array terlebih dahulu.

B. PSEUDOCODE

```
Prosedur heapify(array, n, i):

  paling_besar = i

  kiri = 2 * i + 1

  kanan = 2 * i + 2



  jika kiri < n dan array[kiri] > array[paling_besar]:

    paling_besar = kiri
```

```
    jika kanan < n dan array[kanan] > array[paling_besar]:

        paling_besar = kanan



    jika paling_besar != i:

        tukar(array[i], array[paling_besar])

        heapify(array, n, paling_besar)



Prosedur heapsort(array):

    n = panjang(array)



    # Membangun max heap

    untuk i dari (n // 2) - 1 turun ke 0 lakukan:

        heapify(array, n, i)



    # Melakukan penyortiran

    untuk i dari n - 1 turun ke 0 lakukan:

        tukar(array[0], array[i])  # Pindahkan elemen terbesar ke posisi terakhir yang
belum diurutkan

        heapify(array, i, 0)       # Memanggil heapify untuk memperbaiki struktur heap
pada subarray yang belum diurutkan
```

C. SOURCE CODE

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

            if left < n and arr[left] > arr[largest]:
```

```
            largest = left

        if right < n and arr[right] > arr[largest]:
            largest = right

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

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

        # Membangun max heap
        for i in range(n // 2 - 1, -1, -1):
            heapify(arr, n, i)

        # Melakukan penyortiran
        for i in range(n - 1, 0, -1):
            arr[0], arr[i] = arr[i], arr[0]  # Pindahkan elemen terbesar ke posisi terakhir
    yang belum diurutkan
            heapify(arr, i, 0)              # Memanggil heapify untuk memperbaiki
    struktur heap pada subarray yang belum diurutkan

    # Contoh penggunaan
    arr = [12, 11, 13, 5, 6, 7]
    heapsort(arr)
print("Array yang telah diurutkan:", arr)
```

Hasil Screenshot :

D. ANALISIS KEBUTUHAN WAKTU

1. **Analisis dengan Memperhatikan Operasi/Instruksi:**
    - Fungsi `heapify` memiliki kompleksitas waktu yang bergantung pada tinggi pohon biner maksimum, yang dapat didefinisikan sebagai logaritma basis-2 dari jumlah elemen dalam array, yaitu log(n).
    - Pada setiap panggilan rekursif dalam `heapify`, fungsi tersebut memeriksa dan menukar elemen, yang memerlukan beberapa operasi aritmatika dan perbandingan.
    - Jumlah operasi pada setiap panggilan `heapify` berkisar antara O(1) hingga O(log n), tergantung pada jumlah elemen di bawahnya.

- Dalam `heapsort`, iterasi untuk membangun max heap memiliki kompleksitas waktu O(n).
- Iterasi untuk penyortiran memiliki kompleksitas waktu O(n log n) karena memanggil `heapify` sebanyak n kali dengan kompleksitas O(log n).

2. **Analisis Berdasarkan Jumlah Operasi Abstrak:**
   - Untuk `heapify`, dalam kasus terburuk, kita memiliki log(n) panggilan rekursif, masing-masing memerlukan beberapa operasi.
   - Dalam `heapsort`, membangun max heap memerlukan sekitar n panggilan `heapify`, masing-masing dengan kompleksitas log(n).
   - Penyortiran memerlukan sekitar n panggilan `heapify` dengan kompleksitas log(n).
   - Jadi, jumlah total operasi abstrak adalah sekitar O(n log n).

3. **Analisis Menggunakan Pendekatan Best-case, Worst-case, dan Average-case:**
   - Best-case: Kasus terbaiknya terjadi ketika array sudah merupakan max heap sebelum dimulainya `heapsort`. Dalam hal ini, membangun max heap memerlukan O(n) operasi, dan penyortiran memerlukan O(n log n) operasi. Jadi, kompleksitas totalnya adalah O(n log n).
   - Worst-case: Kasus terburuk terjadi ketika array terurut terbalik. Dalam hal ini, membangun max heap memerlukan O(n) operasi, dan penyortiran memerlukan O(n log n) operasi. Jadi, kompleksitas totalnya adalah O(n log n).
   - Average-case: Kompleksitas rata-rata dari heapsort juga adalah O(n log n). Ini karena dalam banyak kasus, baik pembangunan max heap maupun penyortiran memerlukan waktu yang sama seperti dalam kasus terburuk.[1], [2]
   -

E. REFERENSI

[1] "Analysis_of_Modified_Heap_Sort_Algorithm".

[2] "Performance Analysis of Heap Sort and Insertion Sort Algorithm," *International Journal of Emerging Trends in Engineering Research*, vol. 9, no. 5, pp. 580–586, May 2021, doi: 10.30534/ijeter/2021/08952021.

F. LINK GITHUB

https://github.com/ZidanRaihanzaRafi/Heapsort2