Muhammad Pajar Kharisma Putra

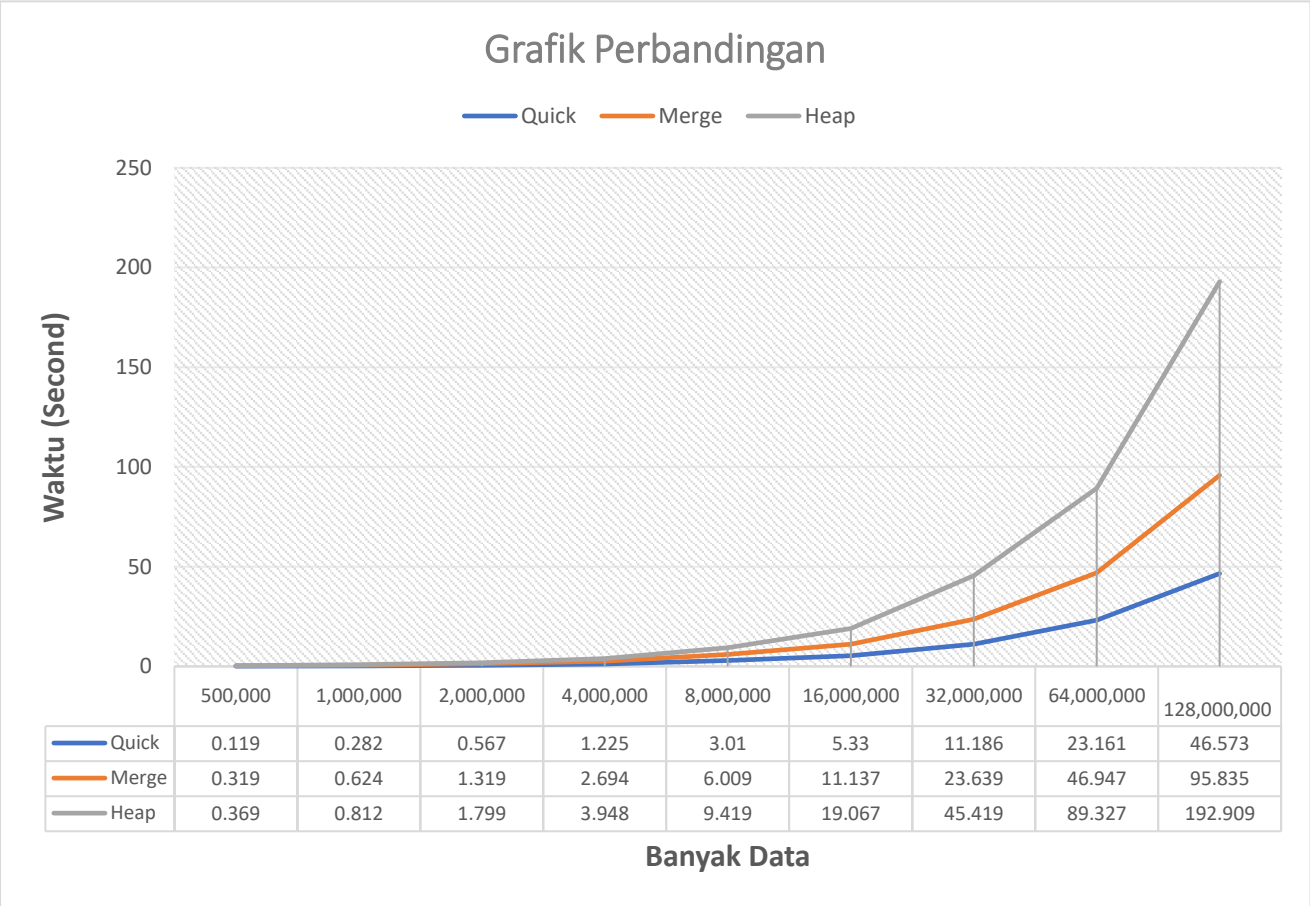
Pra S2 B

629/Pra-MIK/23/2019

A. Tabel perbandingan algoritma Quick Sort, Merge Sort, dan Heap Sort dengan data random.

Banyak data	Lama Proses Algoritma (Detik)		
	Quick	Merge	Heap
500,000	0.119	0.319	0.369
1,000,000	0.282	0.624	0.812
2,000,000	0.567	1.319	1.799
4,000,000	1.225	2.694	3.948
8,000,000	3.01	6.009	9.419
16,000,000	5.33	11.137	19.067
32,000,000	11.186	23.639	45.419
64,000,000	23.161	46.947	89.327
128,000,000	46.573	95.835	192.909

B. Grafik perbandingan.



C. Hasil satu percobaan dengan data yang sudah terurut.

```
Input banyak data : 100000
START SORTING
Waktu Merge Sort : 0.09375 second
Waktu Heap Sort : 0.15625 second
Waktu Quick Sort : 18.3125 second
```

D. Source Code

⇨ Quick Sort

```
int partition (int *arr, int low, int high){
    int pivot = low;
    int i = low;
    int j = high;
    while(j > i){
        if (arr[j] <= arr[pivot]){
            i++;
            swap(arr[i], arr[j]);
        }
        j--;
    }
    swap(arr[i], arr[pivot]);
    return i;
}

void quickSort(int *arr, int low, int high){
    if (low < high){
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

⇨ Merge Sort

```
void merge(int *arr, int l, int m, int r){
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;
    int *L = new int[n1];
    int *R = new int[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2){
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else{
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1){
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2){
        arr[k] = R[j];
        j++;
        k++;
    }
    delete[] L;
    L = NULL;
    delete[] R;
    R = NULL;
}

void mergeSort(int *arr, int l, int r){
    if (l < r){
        int m = l+(r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
```

⇨ **Heap Sort**

```
void heapify(int *arr, int n, int i) {
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

    if (l < n && arr[l] > arr[largest])
        largest = l;

    if (r < n && arr[r] > arr[largest])
        largest = r;

    if (largest != i){
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int *arr, int n){
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i=n-1; i>=0; i--){
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}
```

E. Spesifikasi Hardware

Computer Name:	LISA
Operating System:	Windows 10 Pro 64-bit (10.0, Build 17763)
Language:	English (Regional Setting: English)
System Manufacturer:	LENOVO
System Model:	20235
BIOS:	InsydeH2O Version 03.72.2778CN25WW(V2.03)
Processor:	Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz (4 CPUs), ~2.5GHz
Memory:	12288MB RAM
Page file:	4334MB used, 9710MB available
DirectX Version:	DirectX 12

F. Kesimpulan

a. Quick Sort

- Kompleksitas algoritma quick sort untuk best case dan average case adalah n log n. Sedangkan untuk worst case n^2.
- Worst case quick sort adalah ketika data sudah terurut semua baik ascending maupun descending atau semua elemen data sama.
- Sedangkan best casenya adalah ketika pivot merupakan median dari setiap partisi.

- Algoritma ini hemat memory karena data akan langsung diolah dalam array yang sudah ada, tidak membutuhkan array tambahan lagi.
- Besar kemungkinan partisi tidak stabil (tidak sama besar).

b. Merge Sort

- Algoritma merge sort memiliki kompleksitas yang stabil yaitu n log n baik best case, worst case, ataupun average case.
- Namun algoritma ini memiliki kelemahan yaitu membutuhkan memory yang besar untuk proses merge element array.
- Best case algoritma merge sort adalah ketika data input sudah terurut semua.
- Sedangkan worst case adalah ketika data input terurut silang.
- Contoh input worst case merge sort : 1 3 5 7 9 2 4 6 8 10.
- Merge sort kurang direkomendasikan untuk data dengan jumlah yang sedikit, karena proses kerjanya yang membagi semua element array hingga berjumlah satu. Proses ini akan membutuhkan waktu.

c. Heap Sort

- Heapsort merupakan algoritma sorting yang memanfaatkan heaptree dalam proses pengurutannya.
- Secara umum, kompleksitas dari algoritma ini adalah n log n untuk best case, average case, dan worst case. Namun, karena algoritma ini membutuhkan build heap tree dengan kompleksitas sama yaitu n log n, sehingga keseluruhan kompleksitas algoritma ini adalah 2*(n log n).

- Pada percobaan di atas dengan diberikan data random didapatkan jika quicksort memiliki waktu proses paling cepat diantara semua algoritma, selanjutnya mergesort dan heapsort.

- Namun ketika diberikan data yang sudah terurut, waktu proses quicksort jauh lebih lambat dibanding 2 algoritma lain.

- Mergesort dan heapsort cenderung stabil diberikan data dengan kondisi apapun.

- Mergesort lebih direkomendasikan jika kondisi data input tidak diketahui dan memiliki memory yang cukup besar karena algoritma ini stabil diberikan input data bagaimanapun kondisinya.

- Jika data input besifat random maka quicksort lebih direkomendasikan karena lebih cepat dan lebih hemat memory.