Nama: Alvaro Austin
NPM: 2106752180
Kode Asdos: 3
Kelas: A

Perbandingan Kinerja Two-Pivot-Block-Quicksort dan Merge Sort

Pakta Integritas

Dengan ini saya menyatakan bahwa TE ini adalah hasil pekerjaan saya sendiri.

Daftar Isi

1 Pendahuluan

Algoritma pengurutan (*sorting algorithm*) adalah prosedur atau metode yang digunakan untuk mengatur atau mengurutkan sejumlah elemen data atau item dalam suatu urutan tertentu, seperti urutan numerik. Terdapat berbagai macam algoritma pengurutan yang berbeda, dimana setiap algoritma tersebut memiliki kelebihan dan kekurangannya masing-masing. Dalam tugas eksperimen ini, akan diperlihatkan perbandingan antara dua algoritma pengurutan, yaitu **Two-Pivot-Block-Quicksort** dan **Merge Sort**.

Kedua algoritma tersebut menggunakan algoritma *divide and conquer*. Algoritma divide and conquer adalah paradigma pemecahan masalah dalam ilmu komputer dan matematika yang melibatkan tiga langkah utama:

1. Divide: Masalah yang besar dibagi menjadi submasalah yang lebih kecil. Ini dilakukan dengan cara membagi masalah asal menjadi beberapa submasalah yang lebih kecil dan lebih mudah untuk dipecahkan.
2. Conquer: Submasalah-submasalah yang lebih kecil tersebut dipecahkan atau ”ditaklukkan”. Ini adalah langkah dimana solusi untuk submasalah tersebut ditemukan.
3. Combine: Solusi dari submasalah-submasalah yang lebih kecil digabungkan menjadi solusi untuk masalah asal.

Algoritma divide and conquer sering digunakan untuk memecahkan masalah yang kompleks menjadi masalah yang lebih kecil dan lebih mudah dikelola. Ketika masalah menjadi lebih kecil, solusi untuk masalah tersebut dapat ditemukan dengan lebih mudah, dan kemudian solusi-solusi ini digabungkan kembali untuk mendapatkan solusi akhir. Algoritma *divide and conquer* ini menjadi dasar utama kedua algoritma diatas.

Tujuan utama eksperimen ini adalah untuk memahami kinerja relatif dari kedua algoritma ini dalam berbagai situasi, termasuk pada data yang sudah terurut, data acak, dan data terurut secara terbalik. Hasil eksperimen ini akan memberikan wawasan yang berharga dalam pemilihan algoritma pengurutan yang paling sesuai untuk berbagai aplikasi. Kerangka kerja eksperimen akan melibatkan pembuatan implementasi dari kedua algoritma ini, pengumpulan data performa, dan analisis hasil.

2 Pembahasan

Pada bagian ini, akan dilakukan penjelasan dan perbandingan kedua algoritma Merge Sort dan Two-Pivot-Block-Quicksort yang mendalam. Dalam sisi penjelasan, akan dijelaskan langkah-langkah setiap algoritma, dan dalam tahap perbandingan, akan dilakukan pembandingan kinerja kedua algoritma.

2.1 Merge Sort

Merge sort didefinisikan sebagai algoritma pengurutan yang bekerja dengan membagi sebuah *array* menjadi *subarray* yang lebih kecil, mengurutkan setiap *subarray*, dan kemudian menggabungkan *subarray* yang telah diurutkan kembali bersama untuk membentuk *array* akhir yang sudah diurutkan.

2.1.1 Cara kerja

Merge sort adalah algoritma rekursif yang terus-menerus membagi *array* menjadi dua bagian hingga tidak dapat dibagi lebih lanjut, yaitu saat *array* hanya memiliki satu elemen tersisa (*array* dengan satu elemen selalu terurut). Kemudian *subarray* yang telah diurutkan digabungkan menjadi satu *array* yang terurut. Berikut adalah cara kerja singkat merge sort:

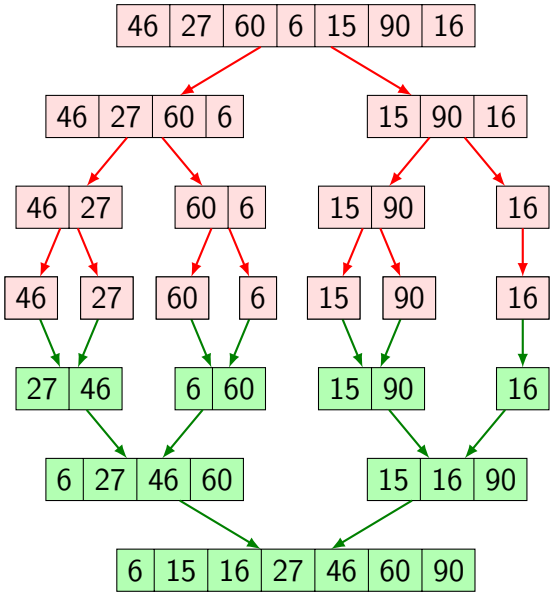

Figure 1: Merge Sort

2.1.2 Algoritma Merge Sort

Algoritma ini mengevaluasi *array* A dimana nilai p merupakan indeks awal dan q merupakan index akhir pada suatu rentang. Pada akhir algoritma ini, *array* A akan menjadi terurut dari elemen terkecil hingga terbesar (*ascending order*). Visualisasi dapat dilihat lagi pada Figure 1. Berikut adalah *pseudocode* algoritma merge sort ini.

Algoritma 2.1 Merge Sort

```
1: function MERGE-SORT(A, p, r)
2:     if p < r then
3:         q ← ⌊(p + r)/2⌋;          ▷ q merupakan titik tengah (midpoint) dari rentang
4:         MERGE-SORT(A, p, q);
5:         MERGE-SORT(A, q + 1, r);
6:         MERGE(A, p, q, r);
```

Dapat dilihat bahwa pada *pseudocode* tersebut, terdapat fungsi *MERGE* yang dipanggil. Fungsi *MERGE* ini berasal dari algoritma Merge pada Algoritma 2.2.

2.1.3 Algoritma Merge

Algoritma ini penggabungan dua bagian terurut dari sebuah *array* menjadi satu bagian terurut. Proses ini melibatkan perbandingan elemen-elemen dari kedua bagian dan penyusunan ulang mereka dalam urutan yang benar dalam *array* asal.

Algoritma 2.2 Merge

```
1:  function MERGE(A, p, q, r)
2:      n1 ← q − p + 1;
3:      n2 ← r − q;
4:      let L[1..n1 + 1] and R[1..n2 + 1] be new arrays;
5:      for i ← 1 to n1 do
6:          L[i] ← A[p + i − 1];
7:      for j ← 1 to n2 do
8:          R[j] ← A[q + j];
9:      L[n1 + 1] ← ∞;
10:     R[n2 + 1] ← ∞;
11:     i ← 1;
12:     j ← 1;
13:     for k ← p to r do
14:         if L[i] ≤ R[j] then
15:             A[k] ← L[i];
16:             i ← i + 1;
17:         else
18:             A[k] ← R[j];
19:             j ← j + 1;
```

2.2 Two-Pivot-Block-Quicksort

Algoritma Two-Pivot-Block-Quicksort adalah variasi dari algoritma *Quicksort* standar, di mana dua pivot digunakan untuk mempartisi *array*. Ide dari Two-Pivot-Block-Quicksort adalah mengambil dua pivot, satu di ujung kiri *array* dan yang kedua, di ujung kanan *array*.

2.2.1 Cara kerja

Pada bagian ini akan dilampirkan cara kerja dan contoh penerapan dari algoritma Two-Pivot-Block-Quicksort. Berikut adalah cara kerja algoritma ini.

1. Dua Pivot dan Block: Pada algoritma ini kita menggunakan 2 pivot ($pivot_1 \leq pivot_2$) dan sebuah *array block*. Kedua pivot masing-masing bernilai 1 dan n dari suatu array utama $A[1..n]$.
2. Daerah: terdapat 3 daerah utama yang terdapat pada algoritma ini. Daerah pertama menandakan elemen lebih kecil dari pivot pertama, daerah kedua menandakan elemen berada diantara pivot pertama dan pivot kedua, dan daerah ketiga menandakan elemen yang lebih besar dari pivot kedua.
3. Block Area: memiliki tujuan untuk memindahkan elemen yang memiliki posisi yang salah keluar dari daerah pivot.

Berikut adalah contoh penerapan dari algoritma Two-Pivot-Block-Quicksort. Asumsikan terdapat sebuah *array* yang dinyatakan sebagai $A = [8,6,7,12,7,5,2,14]$

1. Pemilihan pivot: $pivot_1 = 8$ dan $pivot_2 = 14$.
2. Dilakukan iterasi k pada bagian $[6,7,12]$ dimana dilakukan perbandingan dengan nilai $pivot_2$ dan semua indeks yang elemennya dibawah pivot tersebut, dimasukkan ke array $block = [0,1,2]$. Dilakukan operasi swapping pada elemen pertama sehingga diperoleh $block = [0,1]$

3. Dilakukan iterasi k pada bagian $[7,5,2]$ dimana dilakukan perbandingan dengan nilai $pivot_2$ dan semua indeks yang elemennya dibawah pivot tersebut, dimasukkan ke array *block* yang menghasilkan nilai $[0,1,2]$.
4. Dilakukan swap antara pivot tersebut sehingga pivot berada diantara daerah pemisah tersebut, sehingga diperoleh $A = [2,6,7,7,5,8,12,14]$
5. Partisi A dilakukan secara rekursif berdasarkan pivot pertama dan pivot keduanya berdasarkan masing-masing *subarray*, yaitu $[2,6,7,7,5]$, $[8]$, dan $[]$. Akan tetapi pada 2 *subarray* terakhir tidak perlu karena jumlah elemennya 1.

2.2.2 Algoritma Two-Pivot-Block-Quicksort

Berikut adalah *pseudocode utama* untuk algoritma Two-Pivot-Block-Quicksort. Algoritma ini akan membagi masalah menjadi 3 submasalah yang akan diselesaikan secara rekursif.

Algoritma 2.3 Two-Pivot-Block-Quicksort

```
1:  function TwoPivotBlockQuicksort(A[1..n], l = None, r = None)
2:      if l is None and r is None then
3:          l ← 1;
4:          r ← n;
5:      if l < r then
6:          (i, j) ← TwoPivotBlockPartition(A, l, r);          ▷ Alg 2.4
7:          TwoPivotBlockQuicksort(A, l, i − 1);
8:          TwoPivotBlockQuicksort(A, i + 1, j − 1);
9:          TwoPivotBlockQuicksort(A, j + 1, r);
```

2.2.3 Algoritma Two-Pivot Block Partition

Algoritma ini akan menjadi algoritma partisi utama kita yang menggunakan block dibandingkan algoritma quicksort biasanya.

Algoritma 2.4 Two-Pivot-Block-Partition

```
1:  function TwoPivotBlockPartition(A, l, r)
2:      n ← r − l + 1;
3:      if n <= 1 then return (l, r);
4:      ChoosePivot(A, l, r);                                  ▷ Alg 2.5
5:      p ← A[l];
6:      q ← A[r];
7:      integer block[0, ..., B − 1];
8:      i, j, k ← l + 1, l + 1, 2;
9:      num_<p, num_≤q ← 0;
10:     while k < n do
11:         t ← min(B, n − k);
12:         for c ← 0; c < t; c ← c + 1 do
13:             block[num_≤q] ← c;
14:             num_≤q ← num_≤q + (q ≥ A[k + c + l]);
15:         for c ← 0; c < num_≤q; c ← c + 1 do
16:             Swap A[j + c] and A[k + block[c] + l];
```

```
        k ← k + t ;
        for c ← 0; c < num_{≤q}; c ← c + 1 do
            block[num_{<p}] ← c;
            num_{<p} ← num_{<p} + (p > A[j + c]);
        for c ← 0; c < num_{<p}; c ← c + 1 do
            Swap A[i] and A[j + block[c]];
            i ← i + 1;
        j ← j + num_{≤q};
        num_{<p}, num_{≤q} ← 0;
    Swap A[i − 1] and A[l];
    Swap A[j] and A[r];
    return (i − 1, j);
```

2.2.4 Algoritma Choose Pivot

Berikut adalah algoritma untuk melakukan perbandingan antara kedua pivot dan melakukan penukaran pivot tersebut apabila diperlukan.

Algoritma 2.5 Choose Pivot

```
function ChoosePivot(A, l, r)
    if A[l] > A[r] then
        Swap A[l] and A[r];
```

Untuk implementasi kedua algoritma ini dalam bahasa python, dapat diakses melalui **github saya**. Dalam repository tersebut, saya juga menyediakan kode untuk melakukan generasi dataset agar kode dapat digunakan dengan mudah.

2.3 Perbandingan Algoritma Two-Pivot-Block-Quicksort dan Merge Sort

Pada laporan ini terdapat inti pembahasan yaitu pada perbandingan kedua algoritma antara **Merge Sort** dan **Two-Pivot-Block-Quicksort**. Berikut adalah hasil analisis waktu eksekusi dan penggunaan memori yang saya temukan. Perlu diingat bahwa hasil analisis ini menggunakan mesin lokal dan **bisa saja hasilnya berbeda** pada perangkat lain.

Algorithm	Small			Medium			Big		
	Random	Sorted	Reversed	Random	Sorted	Reversed	Random	Sorted	Reversed
Merge Sort	7.71	7.19	6.84	228.92	245.58	250.17	2447.10	2503.12	2529.41
Two Pivot Quicksort	7.87	157.31	172.84	255.65	69672.63	67477.04	2235.82	4837474.54	5368946.83

Table 1: Tabel Perbandingan Waktu Eksekusi (ms).

Algorithm	Small			Medium			Big		
	Random	Sorted	Reversed	Random	Sorted	Reversed	Random	Sorted	Reversed
Merge Sort	8.60	6.10	4.45	66.03	64.28	64.28	513.60	512.28	512.28
Two Pivot Quicksort	16.72	15.19	15.19	31.08	29.39	29.39	33.79	34.83	34.83

Table 2: Tabel Perbandingan Penggunaan Memori (KB).

Melalui kedua perbandingan hasil waktu ekseksusi dan penggunaan memori, kita dapat melihat bahwa performa kedua algoritma ini bagi dataset yang berstatus *random* memiliki performa waktu eksekusi yang kurang lebih sama. Akan tetapi, dapat dilihat pada tabel 3 bahwa

pengunaan memori kedua algoritma berbeda jauh. Pada algoritma Merge Sort untuk semakin banyak data, maka menggunakan memori yang lebih besar. Namun, pada algoritma Two-Pivot-Block-Quicksort, penggunaan memori tidak bertambah secara signifikan seiring bertambahnya data. Hal ini menandakkan bahwa performa algoritma Two-Pivot-Block-Quicksort ini lebih baik dibandingkan Merge Sort apabila *concern* utamanya berada pada memori. Kasus dimana penggunaan memori menjadi permasalahan yang besar, membuat algoritma Two-Pivot-Block-Quicksort menjadi algoritma yang tepat.

Penggunaan memori yang kecil ini disebabkan oleh properti **2 pivot** dan juga **block**. Quicksort merupakan algoritma yang memanfaatkan pivot untuk menentukkan jumlah partisinya. Berdasarkan *paper* yang diberikan pada referensi, penambahan jumlah pivot meningkatkan *memory access pattern* (MAP) sehingga membuat lebih sedikit akses yang diperlukan dalam mengubah *array* untuk mengurutkan data. Tidak hanya pivot, block juga membantu dalam meningkatkan MAP dengan menambah beberapa perbandingan pada algoritmanya. Sehingga, melalui block, program dapat melakukan tahap pengurutan melalui memori eksternal. Dengan menggunakan block serta 2 pivot pada algoritma Quick Sort pada 257 sampel, kita dapat meningkatkan rata-rata *memory access* sebesar $1.69n\ln n + O(n)$.

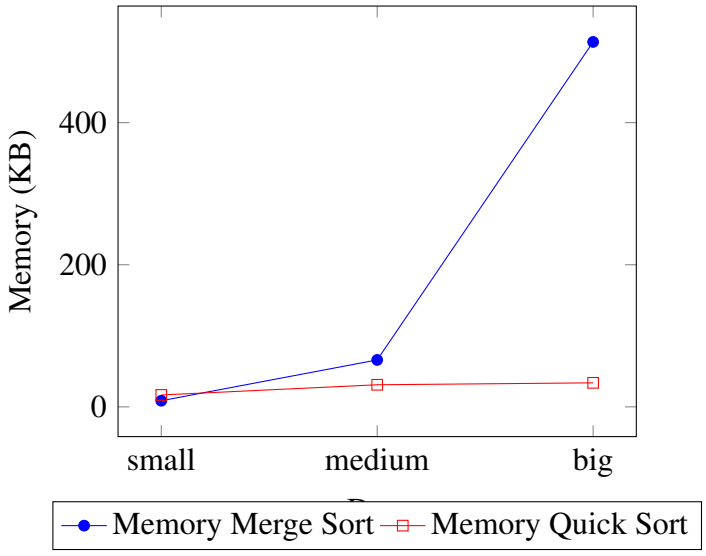

Figure 2: Pengunaan Memori

Disisi lain, algoritma Merge Sort merupakan algoritma yang konsisten dimana untuk status data apapun (*random*, *sorted*, *reversed*) menghasilkan performa yang tidak berbeda jauh. Akan tetapi, hal ini tidak berlaku pada algoritma Two-Pivot-Block-Quicksort. Algoritma ini memiliki karakteristik yang sama dengan algoritma Quick Sort biasa, yaitu tidak berjalan dengan baik apabila data/input yang diberikan sudah dalam keadaan terurut (*ascending* maupun *descending*/*reversed*). Hal ini disebabkan karena pembagian elemen-elemen pada setiap partisi. Apabila data sudah terurut, maka pada setiap partisi, array dibagi menjadi tiga bagian (elemen kurang dari pivot pertama, elemen antara dua pivot, dan elemen lebih besar dari pivot kedua). Namun, karena data sudah terurut maka semua elemen lainnya akan berada di antara dua pivot. Ini berarti kita hanya membagi array menjadi **dua bagian yang tidak seimbang**, bukan tiga.

2.4 Analisis Perbandingan *Running Time*

Berdasarkan hasil yang diperoleh di atas, kita dapat membandingkan kompleksitas *running time* yang diperoleh dengan hasil yang kita peroleh. Kompleksitas *running time* algoritma

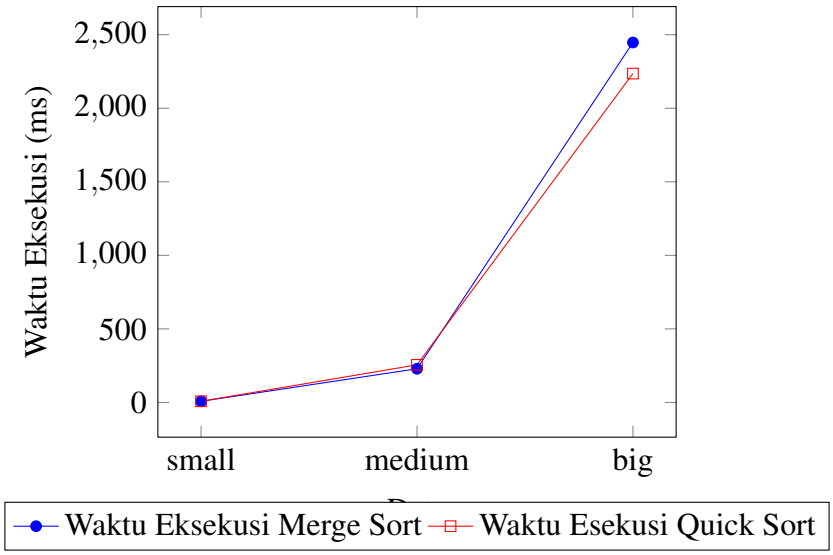

Figure 3: Waktu Eksekusi *Random* Data

Merge Sort adalah $O(n \log n)$ Oleh karena itu, kita bisa membandingkan nilai yang harusnya kita peroleh, misalnya untuk input kecil *random* yang berjumlah 512, perangkat saya berhasil menyelesaikan algoritma mergesort tersebut dalam 7.71 ms. Apabila kita hitung maka kita dapat memperoleh bahwa setiap operasi pada perangkat saya membutuhkan waktu sekitar $1.67x10^{-3}$ ms.

Untuk algoritma Two-Pivot-Block-Quicksort, *average case* yang diperoleh yang biasanya terjadi pada random input adalah $\theta(n \log n)$. Akan tetapi untuk input yang sudah terurut (*ascending* maupun *descending*) adalah $O(n^2)$.

Algorithm	**Small**			**Medium**			**Big**		
	Random	**Sorted**	**Reversed**	**Random**	**Sorted**	**Reversed**	**Random**	**Sorted**	**Reversed**
Merge Sort	7.69	7.69	7.69	177.84	177.84	177.84	1860.56	1860.56	1860.56
Two Pivot Quicksort	7.69	437.78	437.78	177.84	112071.80	112071.80	1860.56	7172595.38	7172595.38

Table 3: Prediksi Running Time (ms).

Dapat dilihat bahwa prediksi *running time* dapat diklasifikasikan sebagai prediksi yang **tepat**. Hal ini disebabkan karena notasi Big-O yang kita gunakan. Untuk merge sort yang memiliki kompleksitas $O(n \log n)$ berlaku ***actual running time*** $\leq$ ***predicted running time*** sesuai dengan tabel diatas. Hal yang sama juga berlaku pada algoritma Two-Pivot-Block-Quicksort dimana kasus *random* yang memiliki kompleksitas $\theta(n \log n)$ dan kasus terurut memenuhi $O(n^2)$ yang memenuhi sifat asimptotik dengan hasil yang diprediksi.

3 Kesimpulan

Algoritma Merge Sort dan Two-Pivot-Block-Quicksort merupakan algoritma pengurutan yang menarik untuk dipelajari. Melalui laporan diatas, kita sudah mengetahui bahwa apabila kita menginginkan konsistensi **dibandingkan** memori, kita dapat memilih algoritma Merge Sort sebagai teknik pengurutan kita. Namun apabila memori menjadi *concern* yang besar, maka algoritma Two-Pivot-Block-Quicksort bisa menjadi pilihan kita. Secara rata-rata, kedua algoritma ini memiliki waktu eksekusi yang sama, namun apabila data yang kita dapatkan adalah data yang sudah terurut, maka penggunaan algoritma Two-Pivot-Block-Quicksort dapat saja

berdampak buruk terhadap performa aplikasi. Hal ini dikarenakan waktu eksekusi algoritma ini memiliki *worst case* $O(n^2)$ sedangkan Merge Sort memiliki *worst case* sebesar $O(n \log n)$. Oleh karena itu penggunaan antara kedua algoritma ini harus dipikirkan baik-baik pada aplikasi yang berkaitan.

4 Referensi

Martin Aumu¨ller and Nikolaj Hass. Simple and Fast BlockQuicksort using Lomuto's Partitioning Scheme. In 2019 *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experi-ments (ALENEX)*, pages 15–26. SIAM, 2019.

GeeksforGeeks. (2023). Dual pivot Quicksort. Retrieved from `https://www.geeksforgeeks.org/dual-pivot-quicksort/`