Laporan Tugas Eksperimen
Nama : Abdul Rafi
NPM : 2106630113
Kode Asdos : 3

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

1. Deskripsi Algortima
    - Two-Pivot-Block-Quicksort

      Algoritma ini merupakan pengenmbangan dari algoritma quicksort biasa. Pada algoritma ini akan dipilih dua buah pivot misal p dan q (p > q). Saat melakukan sorting ini, array kita akan dibagi menjadi tiga bagian.

Elemen bernilai < q	Elemen bernilai >= q dan < p	elemen bernilai >= p

      Kemudian setelah terbagi menjadi 3 bagian tersebut akan dilakukan pembagian lagi untuk masing-masing bagian. Pada algoritma ini juga terdapat tambahan berupa teknik blocking. Teknik ini membagi array menjadi block dengan ukuran tertentu. Hal ini bertujuan untuk mengoptimalkan pengunaan cache. Blocking ini akan lebih terasa pengunaanya apabila kita melakukan sorting pada data yang tidak cukup pada memory (ram) sehingga perlu melibatkan hard disk, dimana pengaksesan pada hard disk jauh lebih lambat dibandingkan dengan memory.

      Pada algortima diatas saya juga menambahkan teknik median of three untuk memilih pivot dari p dan q. nilai p yang saya gunakan merupakan median dari A[L], A[MID], dan A[R]. A merupakan array yang akan dilakukan sorting, L merupakan index paling kiri dari array, R merupakan index paling kanan dari array, dan MID = (L + R) / 2.

      Untuk alur algortima Two-Pivot-Block-Quicksort, misalkan diberikan sebuah array A dengan nilai sebagai berikut A = [12, 1131, 13, 5, 6, 7, 541, 63]. Maka dengan teknik median of three didapatkan p = 12 dan q = 63.

7, 5, 6	12, 13	63, 541, 1131

      Setelah itu kita perlu melakukan sorting kembali untuk [7, 5, 6]. Pada partisi ini didapat p = 6 dan q = 7 sehingga hasilnya menjadi

5	6	7

      Kemudian kembali ke partisi sebelumnya kita perlu melakukan sorting untuk partisi [12, 13]. Didapatkan hasil yaitu [12, 13]. Kemudian dilanjutkan sorting untuk partisi [541,1131]. Didapatkan hasil [541,1131]

      Dengan demikian array berhasil di sort dengan hasil akhir [5, 6, 7, 12, 13, 541, 1131].
    - Merge Sort

Algortima ini akan membagi array menjadi dua terus menerus sampai ukuran array tersebut 1. Ketika ukuran array tersebut sudah kecil kita dapat melakukan sorting berdasarkan array tersebut. Konsep ini dinamakan divide and conquer. Proses pembagian sampai array menjadi kecil ada proses divide. Kemudian ketika disatukan menjadi array besar disebut proses conquer.

Untuk alur algortima Merge sort, misalkan diberikan sebuah array A dengan nilai sebagai berikut A = [12, 1131, 13, 5, 6, 7, 541, 63]. Kita akan membagi array menjadi [12, 1131, 13, 5] dan [6, 7, 541, 63].

Kemudian array [12, 1131, 13, 5] dibagi menjadi [12, 1131] dan [13,5]. kemudian [12,1131] dibagi menjadi [12] dan [1131]. Dari sini kita dapat memulai sorting [12,1131] dan diapat array setelah sort yaitu [12,113].

Array [13,5] dibagi menjadi [13] dan [5]. Kemudian kita dapat melakukan sort array [13,5] menjadi [5, 13]. Kemudian kita dapat menyatukan array [12, 1131] dan [5,13] menjadi [5, 12, 13, 1131].

Dengan begitu kita berhasil melakukan sorting pada setengah bagian pertama. Kita tinggal melakukan hal yang sama untuk setengah bagian berikutnya.

2. Hasil eksperimen dan analisis

Dataset	Merge Sort : Time	Merge Sort : Memory	Two-Pivot-Block-Quicksort : Time	Two-Pivot-Block-Quicksort : Memory
Ukuran 2^9 sorted	1 ms	58 MiB	2.3 ms	58 MiB
Ukuran 2^9 random	0.8 ms	58 MiB	1.9 ms	58 MiB
Ukuran 2^9 reversed	0.7 ms	58 MiB	1.8 ms	58 MiB
Ukuran 2^{13} sorted	15 ms	59 MiB	46 ms	59 MiB
Ukuran 2^{13} random	18 ms	59 MiB	33 ms	59 MiB
Ukuran 2^{13} reversed	15 ms	59 MiB	44 ms	59 MiB
Ukuran 2^{16} sorted	192 ms	62 MiB	441 ms	61 MiB
Ukuran 2^{16} random	209 ms	62 MiB	414 ms	61 MiB
Ukuran 2^{16} reversed	259 ms	62 MiB	462 ms	61 Mib

Berdasarkan data diatas dapat dilihat bahwa pengunaan memory untuk kedua algoritma tersebut tidak terdapat perbedaan yang signifikan. Walaupun kedua

algoritma tersebut memiliki kompleksitas yang sama yaitu O(N log N). Kompleksitas itu diadapat dari banyaknya operasi partisi yang dilakukan pada algoritma merge sort. Pada Two-Pivot-Block-Quicksort akan memiliki kompleksitas O(N log N) apabila pembagian yang dilakukan selalu seimbang.

Dapat dilihat berdasarkan data diatas merge sort berjalan lebih cepat dibandingkan Two-Pivot-Block-Quicksort. Namun, hal ini bisa saja menghasilkan data yang berbeda ketika kita meningkatkan banyaknya dataset sehingga dataset tidak dapat disimpan pada memory.

3. Kesimpulan
   Merge sort berjalan lebih cepat pada dataset yang cukup disimpan pada memory. Hal ini dikarenakan pembagian pada merge sort selalu seimbang. Sedangkan, hal ini tidak selalu terjadi pada Two-Pivot-Block-Quicksort. Namun, Two-Pivot-Block-Quicksort mungkin bisa berjalan lebih cepat dibandingkan Merge Sort karena pada Two-Pivot-Block-Quicksort terdapat teknik Blocking yang digunakan untuk mempercepat pengaksesan cache memory.

4. Psudocode

Github Link : https://github.com/TakeMeGH/DAA-TE1

Two-Pivot-Block-Quicksort

```
def median_of_three(a, low, high):
    mid = (low + high) // 2
    if a[low] > a[mid]:
        a[low], a[mid] = a[mid], a[low]
    if a[mid] > a[high]:
        a[mid], a[high] = a[high], a[mid]
    if a[low] > a[mid]:
        a[low], a[mid] = a[mid], a[low]
    return mid


def block_lomuto2(A, B, low, high):
    mid = median_of_three(A, low, high)
    A[low], A[mid] = A[mid], A[low]

    p = A[low]
    q = A[high]
    block = [0] * B
    i = low + 1
    j = low + 1
```

```
    k = low + 1
    num_less_p = num_leq_q = 0

    while k < high:
        t = min(B, high - k)
        for c in range(t):
            block[num_leq_q] = c
            num_leq_q += q >= A[k + c]

        for c in range(num_leq_q):
            A[j + c], A[k + block[c]] = A[k + block[c]], A[j + c]

        k += t

        for c in range(num_leq_q):
            block[num_less_p] = c
            num_less_p += p > A[j + c]

        for c in range(num_less_p):
            A[i], A[j + block[c]] = A[j + block[c]], A[i]
            i += 1

        j += num_leq_q
        num_less_p = num_leq_q = 0

    A[i - 1], A[low] = A[low], A[i - 1]
    A[j], A[high] = A[high], A[j]

    return i - 1, j

def two_pivot_quicksort_iterative(A, B=100):
    high = len(A) - 1
    stack = [(0, high)]

    while stack:
        low, high = stack.pop()
        if low < high:
            i, j = block_lomuto2(A, B, low, high)

            if j < high:
                stack.append((j + 1, high))
```

```
            if i < j - 1:
                stack.append((i, j - 1))

            if low < i - 1:
                stack.append((low, i - 1))
```

Merge Sort

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```