Nama: Naufal Fikri Hakim  
NPM: 2106750566  
Kode Asdos: 5

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



**TUGAS EKSPLORASI 2**

**Deskripsi Singkat**

Problem yang diberikan di soal merupakan problem set partition. Set partition sendiri merupakan suatu problem dimana terdapat suatu array/list, kemudia kita ingin membagi array/list tersebut menjadi 2 buah subset, dimana jumlah dari nilai elemen-elemen pada kedua subset tersebut sama. Terdapat 2 pendekatan untuk mecari solusi dari masalah tersebut. 2 pedekatan tersebut adalah Dynamic Programming dan Branch and Bound. Dynamic programming sendiri merupakan suatu teknik pemrograman dimana masalah yang ada dipecah menjadi beberapa sub-masalah dan menyimpan solusi dari sub-masalah tersebut. Solusi tersebut disimpan agar apabila ditemukan sub-masalah yang sama, maka program tidak perlu untuk melakukan komputasi lagi dan dapat mempercepat jalannya program. Branch and Bound sendiri merupakan suatu metode dimana metode ini memecah masalah menjadi sub-masalah (branch) dan memberikan batasan (bound) untuk branch yang tidak mungkin menghasilkan hasil yang optimal atau lebih baik.

**Contoh Penerapan**

Misalkan terdapat suatu array yang panjang 7 yang berisikan [3, 4, 1, 2, 9, 7, 6] dan akan diurutkan menggunakan algoritma Clustered Binary Insertion Sort. Berikut merupakan penjelasan pada setiap iterasi.

Iterasi akan dilakukan sejumlah n-1 kali. COP adalah current pointer dan POP adalah position pointer. Iterasi dilakukan mulai dari index pertama karena index ke-0 pasti sudah terurut(sorted).  
  
Inisiasi:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 1 | 2 | 9 | 7 | 6 |

1. COP = 1, POP = 0. Algoritma akan membandingkan terlebih dahulu elemen di COP dan elemen di POP, kemudian algoritma akan memasukkan elemen kedalam index yang sesuai yang didapatkan dari fungsi binary\_loc\_finder. Algoritma akan menaruh angka pada COP (angka 4) di index 1 dan akan meng-update nilai POP menjadi 1.  
   Hasil iterasi 1:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 1 | 2 | 9 | 7 | 6 |

1. COP = 2, POP = 1. Algoritma akan membandingkan nilai 1 dan 4 dan kemudian menjalankan fungsi binary\_loc\_finder untuk menentukan posisi angka 1  
   Hasil iterasi 2:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 2 | 9 | 7 | 6 |

1. COP = 3, POP = 0. Algoritma akan membandingkan nilai 2 dan 1 dan kemudian menjalankan fungsi binary\_loc\_finder untuk menentukan posisi angka 2  
   Hasil iterasi 3:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 9 | 7 | 6 |

1. COP = 4, POP = 1. Algoritma akan membandingkan nilai 9 dan 2, kemudian menjalankan fungsi binary\_loc\_finder untuk menentukan posisi angka 9  
   Hasil iterasi 4:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 9 | 7 | 6 |

1. COP = 5, POP = 4. Algoritma akan membandingkan nilai 7 dan 9, kemudian menjalankan fungsi binary\_loc\_finder untuk menentukan posisi angka 7.  
   Hasil iterasi 4:

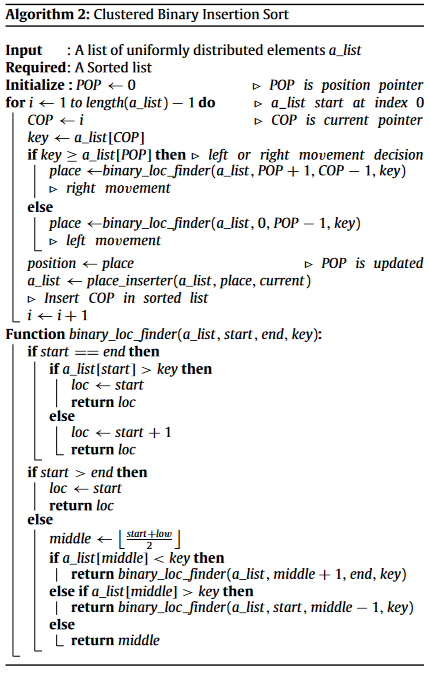
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 7 | 9 | 6 |

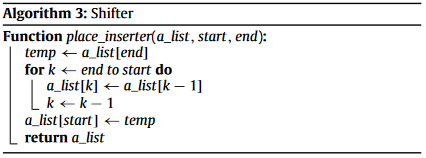
1. COP = 6, POP = 4. Algoritma akan membandingkan nilai 6 dan 7, kemudian menjalankan fungsi binary\_loc\_finder untuk menentukan posisi angka 6.  
   Hasil iterasi 6:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 6 | 7 | 9 |

**Pseudocode dan Tautan Github**

Pseudocode untuk Clustered Binary Insertion Sort:





Pseudocode untuk Randomized QuickSort:

partition(arr[], lo, hi)   
 pivot = arr[hi]  
 i = lo // place for swapping  
 for j := lo to hi – 1 do  
 if arr[j] <= pivot then  
 swap arr[i] with arr[j]  
 i = i + 1  
 swap arr[i] with arr[hi]  
 return i  
partition\_r(arr[], lo, hi)  
 r = Random Number from lo to hi  
 Swap arr[r] and arr[hi]  
 return partition(arr, lo, hi)  
quicksort(arr[], lo, hi)  
 if lo < hi  
 p = partition\_r(arr, lo, hi)  
 quicksort(arr, lo , p-1)  
 quicksort(arr, p+1, hi)

Berikut merupakan link Github yang berisikan dataset yang digunakan dan implementasi dari algoritma BCIS dan QS yang digunakan:

<https://github.com/NaufalFikriHakim/TE1-DAA>

**Hasil Eksperimen dan Analisis**

Berikut ini merupakan tabel yang berisikan perbandingan pengunaan memori(dalam bytes) dan waktu eksekusi(dlaam milliseconds) dari kedua algoritma, yaitu Randomized Quicksort dan Clustered Binary Insertion Sort. Algortima dijalankan terhadap 3 data yang berukuran 200, 2000, dan 20000 yang bersifat sorted, random, dan reversed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Clustered Binary Insertion Sort** | | **Randomized Quick Sort** | |
|  | **Memori** | **Waktu** | **Memori** | **Waktu** |
| **200 Random** | 4476 | 0 | 6504 | 1 |
| **200 Sorted** | 112 | 0 | 80 | 0 |
| **200 Reversed** | 2000 | 0 | 536 | 0 |
| **2000 Random** | 3155 | 2 | 5294 | 1 |
| **2000 Sorted** | 168 | 0 | 1452 | 1 |
| **2000 Reversed** | 2116 | 1 | 1828 | 1 |
| **20000 Random** | 2732 | 34 | 15332 | 2 |
| **20000 Sorted** | 168 | 0 | 2964 | 1 |
| **20000 Reversed** | 4700 | 48 | 5388 | 1 |

Berdasarkan tabel diatas, dapat dilihat bahwa CBIS lebih cepat dalam menyelesaikan data yang sudah sorted, namun lebih lambat dalam menyelesaikan data yang belum sorted dan berukuran besar. Untuk penggunaan memori, dalam mengurutkan data yang besar, CBIS lebih efisien dibandingkan dengan RQS.

CBIS akan memiliki best case scenario apabila terjadi komparasi sebanyak O(n) kali. CBIS juga memiliki time complexity untuk average case sebesar dan worst case sebesar . Sedangkan RQS memiliki time complexity untuk best case sebesar dan worst case sebesar .

**Kesimpulan**

CBIS adalah bentuk modifikasi dari BIS untuk mempercepat waktu yang dibutuhkan dalam menjalankan worst case scenario dari BIS. Apabila dibandingkan dengan randomized quick sort, CBIS lebih baik dalam meng-handle data yang sudah sorted, namun tidak lebih baik dalam meng-handle data yang belum sorted dan berukuran besar. Untuk data yang berukuran besar pula dapat dikatakan bahwa CBIS efisien dalam penggunaan memori daripada randomized quick sort.