**Algoritma Quick Sort dalam Pengurutan Bilangan Integer**

Raihana Salsabila Darma Wijaya, Hariadi Adha Firmansyah, Rendy Putra Pratama

Telkom University

Jl. Telekomunikasi, Jl. Terusan Buah Batu No.01, Sukapura, Dayeuhkolot, Bandung, Jawa Barat 40257

raihanawijaya@student.telkomuniversity.ac.id

### ABSTRAK

### Terdapat banyak algoritma untuk menyelesaikan masalah pengurutan bilangan integer, salah satunya algoritma Quick Sort. Quick Sort adalah algoritma pengurutan yang sangat cepat dengan tipe penyelesaian divide and conquer, sehingga cocok untuk mengurutkan data dalam jumlah besar. Pada makalah ini, kami akan memberikan contoh permasalahan beserta langkah penyelesaiannya. Sebuah algoritma tidak hanya harus benar, tetapi juga harus mangkus atau efisien. Algoritma yang efisien adalah algoritma yang meminimumkan waktu eksekusi dan kebutuhan ruang memori. Maka dari itu, kami juga akan memberikan perhitungan kompleksitas algoritmanya dan analisis perbandingan efisiensi dengan algoritma Brute Force.

**Kata kunci:**Bruteforce, Quicksort, Kata kunci 2, ..., Kata kunci 5.

### 1. PENDAHULUAN

Pada zaman ini, perkembangan teknologi berbanding lurus dengan kemudahan penerimaan suatu informasi atau data. Seiring berjalannya waktu, data yang kita olah semakin besar sehingga dibutuh cara pengolahan data yang efisien dan efektif. Salah satu bentuk pengolahan data yang menjadi permasalahan klasik adalah pengolahan data integer. Pengolahan data integer erat kaitannya dengan pencarian data, dimana dalam pencarian tersebut terdapat proses pemilahan data. Pencarian data yang efektif dan efisien tidak dapat dilepaskan dari faktor keterurutan data. Data yang sudah terurut akan mempermudah dan mempercepat proses pencarian data.

Menurut Astrachan (2003) [2], algoritma pengurutan atau yang biasa dikenal sebagai Sorting Algorithm, telah muncul sejak tahun 1956. Algoritma tersebut dikenal dengan nama Sorting by Exchange. Berbagai macam metode dalam algoritma pengurutan terus ditemukan sampai saat ini. Beberapa contoh algoritma pengurutan adalah Bubble Sort, Insertion Sort, Selection Sort, Shell Sort, Merge Sort, Heap Sort, Quick Sort, dan Bucket Sort. Setiap algoritma pengurutan memiliki pendekatan dan metode yang berbeda-beda dalam menjalankan fungsinya. Secara garis besar, algoritma pengurutan dapat dikelompokkan menjadi dua kategori, yaitu algoritma pengurutan berbasis perbandingan (comparison based) dan tidak berbasis perbandingan (non-comparison based).

Joshi R., Panwar G.R., dan Pathak P. (2013) [4] menyatakan bahwa pada umumnya, algoritma pengurutan yang tidak berbasiskan perbandingan lebih cepat dibandingkan algoritma pengurutan yang berbasiskan perbandingan. Namun disisi lain, dalam buku Data Structures & Algorithms in Java yang dikarang oleh Lafore (2003)[5] menyatakan bahwa algoritma Quick Sort merupakan algoritma pengurutan tercepat secara praktis, dimana algoritma Quick Sort merupakan salah satu contoh dari algoritma yang berbasiskan perbandingan. Maka dari itu kami akan mencoba mengurutkan data bertipe integer dan membandingkan performa yang dihasilkan oleh kedua algoritma tersebut, baik dari sisi waktu maupun penggunaan memori dengan menggunakan algoritma Quicksort.

### 2. QUICK SORT

Quick Sort merupakan suatu algoritma pengurutan data yang menggunakan teknik pemecahan data menjadi partisi-partisi, sehingga metode ini disebut juga dengan nama partition exchange sort. Untuk memulai iterasi pengurutan, pertama-tama sebuah elemen dipilih dari data, kemudian elemen-elemen data akan diurutkan diatur sedemikian rupa.

Quick Sort merupakan algoritma yang ditemukan oleh C. A. R. Hoare pada tahun 1960 dan kemudian dikenalkan secara luas pada tahun 1962. Algoritma ini menggunakan prinsip divide and conquer. Algoritma ini sering dijadikan pilihan karena mudah dan hemat untuk diimplementasikan [2]ht t p://www.personal.kent .edu/~rmuhamma/Algorit hms/MyAlg

orit hms/Sort ing/quickSort .ht m

.

Wakt u akses : 12 Desember

2013, jam 12.14.

Algoritma ini mengambil salah satu elemen secara acak (biasanya dari tengah) yang disebut dengan pivot, lalu menyimpan semua elemen yang lebih kecil di sebelah kiri pivot dan semua elemen yang lebih besar di sebelah kanan pivot. Hal ini dilakukan secara rekursif terhadap elemen di sebelah kiri dan kanannya sampai semua elemen sudah terurut.Dalam algoritma quick sort, pemilihan pivot adalah hal yang menentukan apakah algoritma quick sort tersebut akan memberikan performa terbaik atau terburuk. Berikut beberapa cara pemilihan pivot :

Pivot adalah elemen pertama, elemen terakhir, atau elemen tengah tabel. Cara ini hanya bagus jika elemen tabel tersusun secara acak, tetapi tidak bagus jika elemen tabel semula sudah terurut. Misalnya, jika elemen tabel semula menurun, maka semua elemen tabel akan terkumpul di tabel kanan.

Pivot dipilih secara acak dari salah satu elemen tabel. Cara ini baik, tetapi mahal, sebab memerlukan biaya (cost) untuk pembangkitanprosedur acak. Lagi pula, itu tidak mengurangi kompleksitas waktu algoritma.

Pivot adalah elemen median tabel. Cara ini paling bagus, karena hasil partisi menghasilkan dua bagian tabel yang berukuran seimbang (masing masing ≈ n/2 elemen). Cara ini memberikan kompleksitas waktu yang minimum. Masalahnya, mencari median dari elemen tabel yang belum terurut adalah persoalan tersendiri. Algoritma Quick Sort terdiri dari dua prosedur, yaitu prosedur PARTISI dan prosedur QUICKSORT.

Pseudo-code algoritma ini sebagai berikut [1] :

|  |
| --- |
| **Procedure** Quicksort(input/output A : TabelInt, input I, j: integer)  *{Mengurutkan tabel A[i..j] dengan algoritma Quicksort*  *Masukan : Tabel A[i..j] yang sudah terdefinisi elemen-elemennya.*  *Keluaran : Tabel A[i..j] yang terurut menarik.}*  **Deklarasi**  K : integer  **Algoritma**  if i < j then *{Ukuran (A) > 1}*  *Partisi(A, i, j, k) {dipartisi pada indeks k}*  Qouicsort(A, I, k) *{urut A[i..k] dengan quicksort}*  Quicksort(A, k+1, j) *{urut A[k+1..j] dengan quicksort}*  Endif |

|  |
| --- |
| **Procedure** partisi(input/output A : TabelInt, input I, j: integer,  Output q : integer)  {membagi tabel A[i..j] menjadi uptabel A[i..q] dan A[q+1..j]  Masukan : Tabel A[i..j] yang sudah terdefinisi harganya.  Keluaran : uptabel A[i..q] lebih kecil dari elemen tabel A[q+1..j]}  **Deklarasi**  Pivot, temp : integer  **Algoritma**  Pivot🡨A(i+j) div 2 {pivot = elemen tengah}  p🡨i  q🡨j    repeat  while Ap < pivot do  p🡨p+1  endwhile  {Ap >= pivot}  While Aq > pivot do  q🡨q-1  Endwhile  {Aq <= pivot}  If p<q then  {pertukaran Ap dengan Aq}  Temp🡨 Ap  Ap🡨Aq  Aq🡨temp  {tentukan awal pemindaian berikutnya}  p🡨p + 1  q🡨q – 1  endif |

* **Keunggulan Quick sort:**

Secara umum memiliki kompleksitas O(n log n).

Algoritmanya sederhana dan mudah diterapkan pada berbagai bahasa pemrograman dan arsitektur mesin secara efisien. Dalam prakteknya adalah yang tercepat dari berbagai algoritma pengurutan dengan perbandingan, seperti mergesort dan heapsort.

Melakukan proses langsung pada input (in-place) dengan sedikit tambahan memori.

Bekerja dengan baik pada berbagai jenis input data (seperti angka dan karakter).

* **Kekurangan Quick sort**

Sedikit kesalahan dalam penulisan program membuatnya bekerja tidak beraturan (hasilnya tidak benar atau tidak pernah selesai).

Memiliki ketergantungan terhadap data yang dimasukkan, yang dalam kasus terburuk memiliki kompleksitas O(n2).

Secara umum bersifat tidak stabil, yaitu mengubah urutan input dalam hasil akhirnya (dalam hal inputnya bernilai sama).

Pada penerapan secara rekursif (memanggil dirinya sendiri) bila terjadi kasus terburuk dapat menghabiskan stack dan memacetkan program.

Pada bahasa pemrograman, quicksort ada dalam pustaka stdlib.h untuk bahasa C, dan class TList dan TStringList dalam Delphi (Object Pascal) maupun FreePascal.

**3. RANCANGAN METODE DAN HASIL**

Dalam hal ini kami memiliki angka sebagai berikut

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |

* Langkah pertama adalah tentukan pivotnya. Dalam hal ini adalah kami memilih angka 7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

* Kemudian buat partisi masing masing angka sebelah kiri dan kanan

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
| partisi 1[i] |  |  | pivot |  | Partisi 2[j] |  |

* Kemudian gunakan algoritma quicksort yang ada diatas. Jika angka lebih kecil dari pivot maka akan diletakan sebelah kiri dan jika lebih besar maka letakan disebelah kanan. Langkah pertama adalah dibandingkan angka 9 dengan pivot apakah lebih kecil atau lebih besar

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ▼ |  |  | ▼ |  |  |  |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |
| --- |
| 7 |
| pivot |

* Karena angka 9 lebih besar maka letakan angka 9 setelah pivot.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |
| --- | --- |
| 7 | 9 |
| pivot |  |

* Lanjut ke angka 4, bandingkan angka 4 dengan pivot

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ▼ |  | ▼ |  |  |  |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |
| --- | --- |
| 7 | 9 |
| pivot |  |

* Karena angka 4 lebih kecil dari 7 maka posisi tetap

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |
| --- | --- |
| 7 | 9 |
| pivot |  |

|  |
| --- |
| 4 |
|  |

* Lanjut ke angka 2. Cek apakah angka 2 lebih kecil atau lebih besar dari pivot

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | ▼ | ▼ |  |  |  |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |
| --- | --- |
| 7 | 9 |
| pivot |  |

|  |
| --- |
| 4 |
|  |

Karena angka 2 lebih kecil dari pivot maka letaknya tetap

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |
| --- | --- |
| 7 | 9 |
| pivot |  |

|  |  |
| --- | --- |
| 4 | 2 |
|  |  |

* Bandingkan pivot dengan angka 10. Cek angka 10 lebih besar atau lebih kecil dari pivot.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | ▼ | ▼ |  |  |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |
| --- | --- |
| 7 | 9 |
| pivot |  |

|  |  |
| --- | --- |
| 4 | 2 |
|  |  |

* Karena angka 10 lebih besar maka posisi tetap sebelah kanan

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |  |
| --- | --- | --- |
| 7 | 9 | 10 |
| pivot |  |  |

|  |  |
| --- | --- |
| 4 | 2 |
|  |  |

* Lanjut ke angka 1. Cek angka 1 lebih kecil atau lebih besar dari pivot.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | ▼ |  | ▼ |  |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |  |
| --- | --- | --- |
| 7 | 9 | 10 |
| pivot |  |  |

|  |  |
| --- | --- |
| 4 | 2 |
|  |  |

* Karena lebih kecil maka pindah ke sebelah kiri pivot.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |  |
| --- | --- | --- |
| 7 | 9 | 10 |
| pivot |  |  |

|  |  |  |
| --- | --- | --- |
| 4 | 2 | 1 |
|  |  |  |

* Lanjut ke angka 5. Cek apakah angka 5 lebih kecil atau lebih besar dari angka pivot.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | ▼ |  |  | ▼ |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |  |
| --- | --- | --- |
| 7 | 9 | 10 |
| pivot |  |  |

|  |  |  |
| --- | --- | --- |
| 4 | 2 | 1 |
|  |  |  |

* Karena angka 5 lebih kecil dari pivot, maka pindah ke sebelah kiri pivot,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 2 | 7 | 10 | 1 | 5 |
|  |  |  | pivot |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | 2 | 1 | 5 | 7 | 9 | 10 |
|  |  |  |  | pivot |  |  |

* Setelah itu masuk ke dalam partisi baru. Sampai sini proses belum selesai

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | 2 | 1 | 5 | 7 | 9 | 10 |
|  | **partisi 1[i]** |  |  |  | Partisi 2[j] |  |

* Tentukan pivot untuk masing-masing partisi

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 2 | 1 | 5 |
| pivot |  |  |  |

|  |  |  |
| --- | --- | --- |
| 7 | 9 | 10 |
|  |  | pivot |

* Perbandingan untuk pivot pertama. Angka 2. Cek apakah angka 2 lebih kecil atau lebih besar dari pivot

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ▼ | ▼ |  |  |  |  |  |
| 4 | 2 | 1 | 5 | 7 | 9 | 10 |
| pivot |  |  |  |  |  | pivot |

|  |
| --- |
| 4 |
| pivot |

|  |
| --- |
| 10 |
| pivot |

* Karena angka 2 lebih kecil dari pivot maka pindahkan ke kiri pivot

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | 2 | 1 | 5 | 7 | 9 | 10 |
| pivot |  |  |  |  |  | pivot |

|  |  |
| --- | --- |
| 2 | 4 |
|  | pivot |

|  |
| --- |
| 10 |
| pivot |

* Lanjut ke angka 1. Cek apakah angka lebih besar atau lebih kecil dari pivot.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ▼ |  | ▼ |  |  |  |  |
| 4 | 2 | 1 | 5 | 7 | 9 | 10 |
| pivot |  |  |  |  |  | pivot |

|  |  |
| --- | --- |
| 2 | 4 |
|  | pivot |

|  |
| --- |
| 10 |
| pivot |

* Karena angka 1 lebih kecil maka pindahkan disebelah kiri angka pivot

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | 2 | 1 | 5 | 7 | 9 | 10 |
| pivot |  |  |  |  |  | pivot |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 4 |
|  |  | pivot |

|  |
| --- |
| 10 |
| pivot |

* Lanjut ke angka 5. Cek apakah angkanya lebih besar atau lebih kecil dari pivot.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ▼ |  |  | ▼ |  |  |  |
| 4 | 2 | 1 | 5 | 7 | 9 | 10 |
| pivot |  |  |  |  |  | pivot |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 4 |
|  |  | pivot |

|  |
| --- |
| 10 |
| pivot |

* Karena lebih besar maka posisinya tetap. Untuk partisi pertama selesai

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | 2 | 1 | 5 | 7 | 9 | 10 |
| pivot |  |  |  |  |  | pivot |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 4 | 5 |
|  |  | pivot |  |

|  |
| --- |
| 10 |
| pivot |

* Ulangi langkah – langkah seperti sebelumnya untuk pivot partisi ke 2 dan hasil akhir dari quick sort ini adalah seperti ini :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 5 | 7 | 9 | 10 |

Pada quick sort, sebuah elemen diambil untuk dijadikan pivotdan memindahkan elemen dengan nilai lebih kecil ke sebelah kiri pivot. Elemen dengan nilai yang lebih besar akan diletakkan pada sebelah kanan pivot.Waktu yang dibutuhkan untuk menyusun kembali arrayadalah sebesar Cn dengan C adalah sebuah konstanta. Arraydibagi dua berdasarkan letak pivotyakni berukuran k dan n-k. Kedua sub-array tersebut juga butuh untuk di urutkan. Dengan demikian didapat sebuah persamaan

|  |
| --- |
| *T(n) = T(n – k) + T(k) + Cn* (1) |

T(n) adalah waktu yang dibutuhkan algoritma ini untuk mengurutkan n buah elemen

1. **Worst Case Anaysis**

Worst case terjadi apabila pivot yang dipilih yernyata merupakan elemen terakhir dari array. Dengan demikian didapat dua array berukuran *1* dan *n – 1.* Oleh karena itu dari persamaan(1) didapat

|  |
| --- |
| *T(n) = T(n – 1) + T(1) + Cn* (2) |

Dari persamaan (2) didapat rekurens

|  |
| --- |
| *T(n) = T(n – 1) + T(1) + Cn =*  *[T(n – 1) + T(1) + C(n – 1)] + T(1) + Cn* (3) |

Setelah dilakukan iterasi pada persamaan (2) hingga iterasi ke-i, maka didapat

|  |
| --- |
| *T(n) = T(n – i) + iT(1) + C* (4) |

Rekurens ini hanya akan terjadi sampai *i = n –1.* Oleh karena itu dengan mensubtitusi *i = n – 1* ke persamaan (4) maka akan didapat

|  |
| --- |
| *T(n) = T(1) + (n – 1)T(1) + C = nT(1) + C(n(n – 2) – )* (5) |

Dari persamaan (5), dapat ditarik kesimpulan bahwa algoritma quick sort dalam kondisi worst case memiliki kompleksitas waktu asimptotik sebesar O(n2) [1].

1. **Best Case Analiysis**

Best case pada algoritma quick sort didapat apabila nilai pivot yang diambil tepat membagi array menjadi dua sub – array dengan jumlah elemen yang sama. Dengan kata lain didapat *k = n/2* dan *n – k = n/2.* Dari persamaan (1) didapat rekurensi

|  |
| --- |
| *T(n) = 2T () + Cn = 2(2T () + ()) + Cn*  (6) |

Dengan menajabarkan rekurensi pada persamaaan (6) hingga iterasi ke-k maka didapat

|  |
| --- |
| *T(n) = 2kT () + kCn* (7) |

Iterasi ini akan berhenti sampai nilai k = log n. dengan mensubsitusi k = log n ke persamaan (7) maka didapat

|  |
| --- |
| *T(n) = nT(1) + Cn log n* (8) |

Dari persamaan (8) didapat nilai kompeksitas waktu asimptotik O (n log n)[1].

**4. KESIMPULAN**

dari analisis yang dilakukan untuk menemukan besarnya kompleksitas waktu algoritma dan kompleksitas waktu asimtotik tiap algoritma yang telah dilakukan sebelumnya, dapat dibentuk sebuah tabel untuk menampilkan nilai kompleksitas waktu asimtotik untuk setiap algoritma.

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritma** | **Best Case** | **Average Case** | **Worst Case** |
| Quick sort | O(n log n) | O(n log n) | O(n2) |
| Bubble sort  (brute force) | O(n) | O(n2) | O(n2) |

Quick sort cocok digunakan untuk mengolah data yang tidak terlalu besar. Untuk mengolah data yang besar, disarankan untuk menggunakan merge sort atau heap sort dikarenakan nilai kompleksitas waktu asimtotiknya sebesar O(n log n). bubble sort, insertion sort, dan selection sort tidak cocok digunakan baik untuk data yang besar maupun data kecil karena kompleksitas waktunya masuk kedalam kelompok waktu kuadratik (O(n2)).

**REFERENSI**

[1] Nama, “Judul Makalah”, Nama jurnal, Volume, Nomor, Tahun terbit, halaman.

[2] Nama, “Judul Buku”, Penerbit, Tahun terbit.

<https://id.wikipedia.org/wiki/Quicksort#Ragam-ragam>

<http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2008-2009/Makalah2008/Makalah0809-019.pdf>

<http://dtugasalgoritma.blogspot.com/2010/12/quick-sort-algoritma.html>

<https://docplayer.info/35494326-Komparasi-algoritma-quicksort-dan-bucket-sort-pada-pengurutan-data-integer.html>

<http://onophp.blogspot.com/2018/11/quick-sort-pengertian-agoritma-dan.html>