STRUKTUR DATA

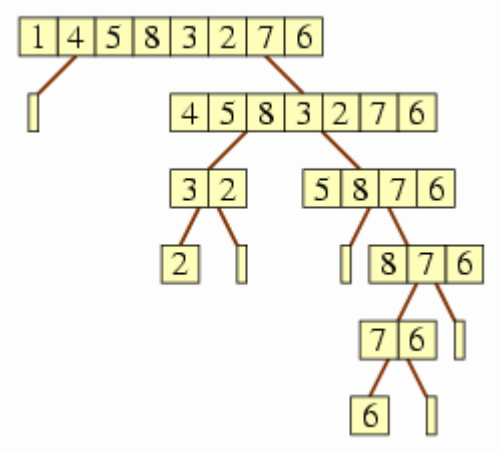
Nama : Sulfikar

Npm : 2013020076

STMIK Handayani Makassar

Pengertian Quick Sort

Algoritma sortir yang efisien yang ditulis oleh C.A.R. Hoare pada 1962. Dasar strateginya adalah "memecah dan menguasai". Quicksort dimulai dengan menscan daftar yang disortir untuk nilai median. Nilai ini, yang disebut tumpuan (pivot), kemudian dipindahkan ke satu sisi pada daftar dan butir-butir yang nilainya lebih besar dari tumpuan di pindahkan ke sisi lain.



Algoritma Quick Sort

DIVIDE

Memilah rangkaian data menjadi dua sub-rangkaian A[p…q-1] dan A[q+1…r] dimana setiap elemen A[p…q-1] adalah kurang dari atau sama dengan A[q] dan setiap elemen pada A[q+1…r] adalah lebih besar atau sama dengan elemen pada A[q]. A[q] disebut sebagai elemen pivot. Perhitungan pada elemen q merupakan salah satu bagian dari prosedur pemisahan.

CONQUER

Mengurutkan elemen pada sub-rangkaian secara rekursif Pada algoritma quick sort, langkah "kombinasi" tidak di lakukan karena telah terjadi pengurutan elemen – elemen pada sub array.

Analisis Algoritma QuickSort

Setiap elemen yang akan disort selalu diperlakukan secara sama di sini, diambil salah satu elemen, dibagi menjadi 3 list, lalu ketiga list tersebut disort dan digabung kembali. Contoh kode di atas menggunakan 3 buah list, yaitu yang lebih besar, sama dan lebih kecil nilainya dari pivot. Untuk membuat lebih efisien, bisa digunakan 2 buah list dengan mengeliminasi yang nilainya sama (bisa digabung ke salah satu dari 2 list yang lain). Kasus terburuk dari algoritma ini adalah saat dibagi menjadi 2 list, satu list hanya terdiri dari 1 elemen dan yang lain terdiri dari n-2 elemen. Untuk kasus terburuk dan kasus rata-rata, algoritma ini memiliki kompleksitas sebesar O(n log n). Jumlah rata-rata perbandingan untuk quick sort berdasarkan permutasinya dengan asumsi bahwa nilai pivot diambil secara random adalah :

$$C(n) = n - 1 + \frac{1}{n}\sum_{i=0}^{n-1}(C(i) + C(n-i-1)) = 2n \ln n = 1.39n \log_2 n.$$

Lalu bagaimana cara menentukan pivot sendiri? Kasus terbaik yang diharapkan diilustrasikan sebagai berikut:
Bagi sebuah list menjadi 4 buah. Lalu pilih 2 buah list sedemikian rupa sehingga setiap elemennya lebih besar dari 25 % elemen terkecil dan lebih kecil dari 25% elemen terbesar. Bila nilai pivot yang dipilih secara konstan terambil dari nilai ini maka hanya diperlukan pembagian list sebanyak 2log2n kali.Biladibandingkan dengan merge sort, quick sort memiliki keuntungan di kompleksitas waktu sebesar Θ(log n), dibanding dengan merge sort sebesar Θ(n). namun quick sort tidak mampu membandingkan linked list sebaik merge sort, karena ada kemungkinan pemilihan pivot yang buruk. Selain itu pada linked list merge sort memerlukan ruang yang lebih sedikit. Berdasarkan analisis tersebut quick sort termasuk algoritma sorting yang cukup baik, namun kita pun harus bisa memilih nilai pivot yang baik agar penggunaannya bisa optmal.

Pseudocode Quick Sort

Procedure Partisi (input/output: a : array[1..n] of integer, input i , j : integer, output q : integer)

{Membagi tabel a[i..j] menjadi subtabel a[i..q] dan a[q+1..j]. Keluaran upatabel a[i..q] dan subtabel a[q+1..j]. Sedemikian sehingga elemen tabel 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 a[p] < pivot do

p <- p + 1

endwhile

{ Ap >= pivot }

while a[q] > pivot do

q <- q – 1

endwhile

{ Aq >= pivot }

if (p _ q) then

{ pertukarkan a[p] dengan a[q]}

temp <- a[p]

a[p] <- a[q]

a[q] <- temp

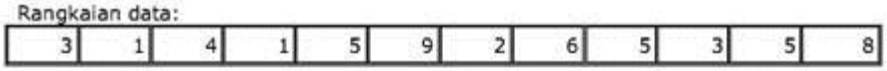
{ tentukan awal pemindaian berikutnya}

p <- p+ 1

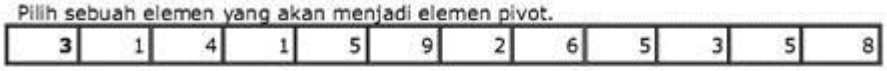
```
q <- q – 1
endif
until p > q
```

Contoh Quick Sort

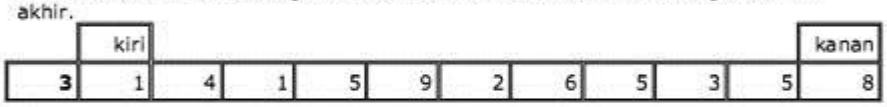
Rangkaian data:



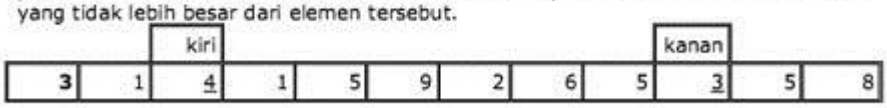
Pilih sebuah elemen yang akan menjadi elemen pivot.



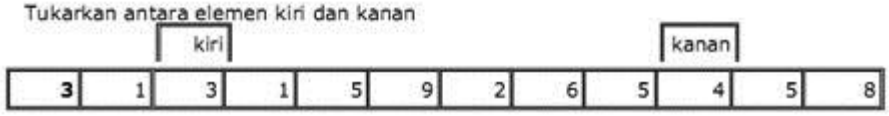
Inisialisasi elemen kiri sebagai elemen kedua dan elemen kanan sebagai elemen akhir.



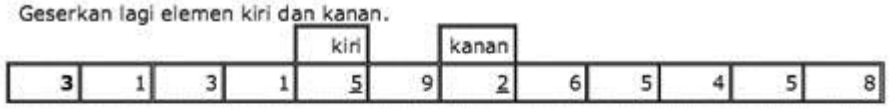
Geser elemen kiri kearah kanan sampai ditemukan nilai yang lebih besar dari elemen pivot tersebut. Geser elemen kanan ke arah kiri sampai ditemukan nilai dari elemen yang tidak lebih besar dari elemen tersebut.



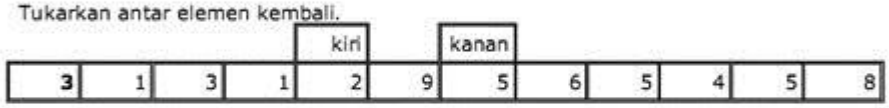
Tukarkan antara elemen kiri dan kanan



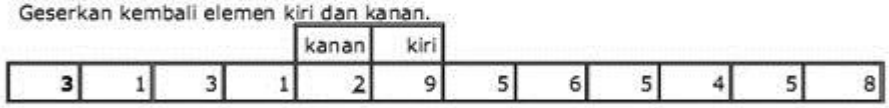
Geserkan lagi elemen kiri dan kanan.



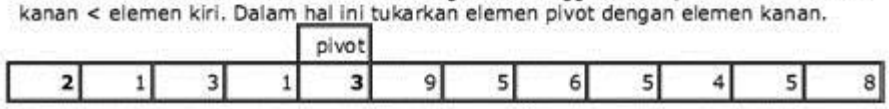
Tukarkan antar elemen kembali.



Geserkan kembali elemen kiri dan kanan.



Terlihat bahwa titik kanan dan kiri telah digeser sehingga mendapatkan nilai elemen kanan < elemen kiri. Dalam hal ini tukarkan elemen pivot dengan elemen kanan.



Kelebihan dan Kekurangan Algoritma Quick Sort

Beberapa hal yang membuat quick sort unggul:

- 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 merge sort dan heap sort.
- Melakukan proses langsung pada input (in-place) dengan sedikit tambahan memori.
- Bekerja dengan baik pada berbagai jenis input data (seperti angka dan karakter).

Namun terdapat pula 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 stable, 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.

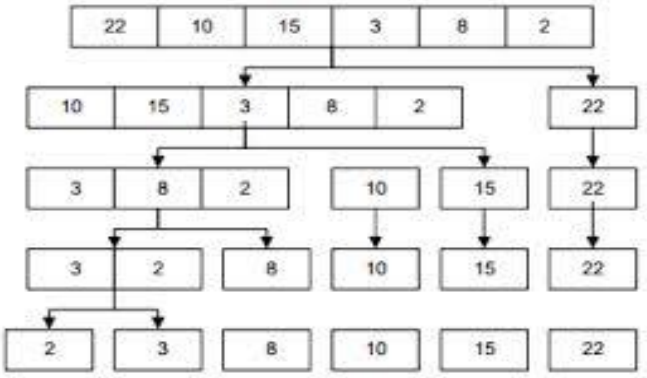
Implementasi Quic Sort Menggunakan C/C++

```
1  Partition(A, p, r)
2  x = A[p]; //pivot=elemen posisi pertama
3  i = p ; //inisialisasi
4  j = r ;
5  repeat
6    while(A[j] > x)
7      j--;
8      while(A[i] < x)
9         i++;
10    if (i < j){
11      Swap(A, i, j);
12      j--;
13      i++
14      }
15    else
16      return j;
17 until i >= j
```

SHELL SORT

Metode Shell (Shell Sort)

Metode ini disebut juga dengan metode pertambahan menurun (diminishing increment). Metode ini dikembangkan oleh Donald L. Shell pada tahun 1959, sehingga sering disebut dengan Metode Shell Sort. Metode ini mengurutkan data dengan cara membandingkan suatu data dengan data lain yang memiliki jarak tertentu, kemudian dilakukan penukaran bila diperlukan



Algoritma Shell Sort

```
Banyak = N
range = banyak / 2
while range <> 0
  counter = 1
  target = banyak-range
  while counter<=target
    kiri = counter
    selesai = false
    while selesai=false
      kanan = kiri+range
if item (kiri)>= item
  (kanan)
     selesai = true
  else
     swap item (kiri) dan
     item (kanan)
  if kiri > range
     kiri = kiri - range
  else
     selesai = true
  end_if
   end_if
   end_while
   counter = counter + 1
   end_while
  range = range / 2
end_while
```

Kelebihan dan Kekurangan Shell Sort

- Kelebihan
  1. Algoritma ini sangat rapat dan mudah untuk diimplementasikan.
  2. Operasi pertukarannya hanya dilakukan sekali saja.
  3. Waktu pengurutan dapat lebih ditekan.
  4. Mudah menggabungkannya kembali.
  5. Kompleksitas selection sort relatif lebih kecil.
- Kekurangan
  1. Membutuhkan method tambahan.
  2. Sulit untuk membagi masalah.

Contoh Program

Contoh Program Kedua (Mengurutkan Data Tipe Angka)

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int arr[30];
 int i,j,k,tmp,num;
 printf("Masukan Banyaknya Elemen :");
 scanf("%d", &num);
 for(k=0; k<num; k++)
 {
  printf("\nMasukkan %d Nilai : ",k+1);
  scanf("%d",&arr[k]);
 }
for(i=num/2; i>0; i=i/2)
 {
  for(j=i; j<num; j++)
  {
   for(k=j-i; k>=0; k=k-i)
   {
     if(arr[k+i]>=arr[k])
                {
       break;
                }
     else
     {
       tmp=arr[k];
```

```
            arr[k]=arr[k+i];
            arr[k+i]=tmp;
        }
      }
    }
  }
 printf("\n**** Hasil Shell Sort ****\n");
 for(k=0; k<num; k++)
     printf("%d\t",arr[k]);
 getch();
 return 0;
}
```