Algoritme dan Pemrograman

Kuliah #12
- Sorting
- Searching

Sorting

- Mengurutkan data berdasarkan kunci tertentu.
- Jenis sorting:
  - Ascending (menaik)
  - Descending (menurun)
- Manfaat : mempercepat dan memudahkan akses data
- Banyak algoritme sorting dikembangkan
  - Mempercepat proses
  - Efisiensi penggunaan sumberdaya

Contoh

- Masukan (input):
  5 2 4 6 1 3
- Keluaran (output):
  1 2 3 4 5 6 (ascending)
  6 5 4 3 2 1 (descending)

Data untuk sorting

- Anggaplah kita mempunya array data yang terdiri dari beberapa kolom, salah satunya adalah key dengan tipe int sbb:

```
struct list {
   int key;
   int val;  // contoh elemen lainnya
   ... // dan seterusnya
};
typedef struct list LIST;
```

- Akan dilakukan sorting berdasarkan nilai key secara ascending.

Fungsi pendukung

- Menukar record t1 dan t2:

```
void tukar(LIST *t1, LIST *t2) {
   LIST t=*t1;
   *t1=*t2;
   *t2=t;
}
```

- Menampilkan n record data

```
void printlist(LIST t[], int n) {
   int i;
   for (i=0; i<n; i++)
      printf("%d %d\n", t[i].key, t[i].val);
}
```


Algoritme #1: BUBBLE SORT

- Baca elemen array dari kiri ke kanan, bandingkan nilai key yang bersebelahan. Jika dua nilai posisinya tidak sesuai, tukar tempatnya. Ulangi prosedur ini sampai semua elemen terurut.
- Keuntungan:
  - Sederhana dan mudah diimplementasikan
  - Cepat jika data masukan sudah terurut
- Kelemahan:
  - Secara umum membutuhkan waktu proses lebih lama

Algoritme #1: BUBBLE SORT

```
void bubbleSort(LIST x[], int n)
{
   int i,j;
   for(i=0;i<n-1;i++){
      for(j=0;j<n-(i+1);j++){
         if (x[j].key > x[j+1].key) {
            tukar(&x[j],&x[j+1]);
         }
      }
   }
}
```


Algoritme #2: SELECTION SORT

- Memilih nilai terkecil dari key dalam array dan tempat record pada posisi yang sesuai berdasarkan nilai key.
- Keuntungan:
  - Sederhana dan mudah diimplementasikan
- Kelemahan:
  - Kurang efisien untuk data berukuran besar

Algoritme #2: SELECTION SORT

```
void selectionSort(LIST t[], int n) {
   int i, j;
   int min;

   for (i=0; i<n; i++) {
      min = i;
      for (j=i+1; j<n; j++) {
         if (t[j].key < t[min].key)
            min = j;
       }
       tukar(&t[i], &t[min]);
   }
}
```


Algoritme #3: INSERTION SORT

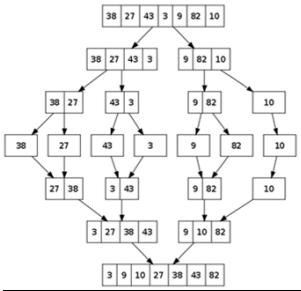
- Baca elemen dalam array dari kiri ke kanan, dan tempatkan pada posisi yang sesuai berdasarkan nilai key.
- Keuntungan:
  - Sederhana dan mudah diimplementasikan
- Kelemahan:
  - Kurang efisien untuk data berukuran besar

Algoritme #3: INSERTION SORT

```
void insertionSort(LIST t[], int n)
{
  int i, j;
  LIST index;

  for (i=1; i<n; i++) {
    index = t[i];
    j = i;
    while ((j>0) && (t[j-1].key>index.key))
    {
      t[j] = t[j-1];
      j = j-1;
    }
    t[j] = index;
  }
}
```


Algoritme #4: MERGE SORT



Algoritme #4: MERGE SORT

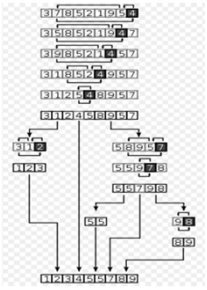
```
void mergeSort(LIST t[], int n)
{
   ms(t, 0, n-1);
}

void ms(LIST t[], int left, int right)
{
   int mid;
   if (left<right) {
      mid=(left+right)/2;
      ms(t, left, mid);
      ms(t, mid+1, right);
      merge(t, left, right, mid);
   }
}
```


Algoritme #4: MERGE SORT

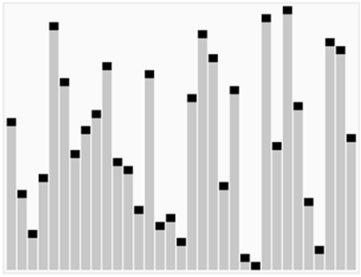
```
merge(LIST t[], int left, int right, int mid) {
   LIST c[SIZE];
   int i=left, j=mid+1, k=left;
   while ((i<=mid)&&(j<=right)) {
      if(t[i].key<t[j].key) {
         c[k]=t[i]; k++; i++;
      } else {
         c[k]=t[j]; k++; j++;
      }
   }
   while (i<=mid) {
      c[k]=t[i]; k++; i++;
   }
   while (j<=right) {
      c[k]=t[j]; k++; j++;
   }
   for(i=left;i<k;i++) t[i]=c[i];
}
```


Algoritme #5: QUICK SORT



- Menerapkan strategi divide and conquer untuk membagi list menjadi dua sublist
- Tahapan
  - Ambil elemen, disebut pivot, dari list
  - Posisikan pivot sehingga elemen sebelah kiri lebih kecil dari pivot, dan elemen sebelah kanan lebih besar dari pivot
  - Lakukan hal yang sama untuk sublist sebelah kiri dan kanan pivot secara rekursif.

Algoritme #5: QUICK SORT (animation)



Algoritme #5: QUICK SORT

```
void quickSort(LIST t[], int n)
{
   qs(t, 0, n-1);
}
```


Algoritme #5: QUICK SORT

```
void qs(LIST t[], int left, int right)
{
    int i, j;
    LIST x, y;

    i = left; j = right;
    x = t[(left+right)/2];

    do {
       while((t[i].key<x.key) && (i<right)) i++;
       while((x.key<t[j].key) && (j>left)) j--;

       if(i <= j) {
          tukar(&t[i], &t[j]);
          i++; j--;
       }
    } while(i<=j);

    if (left<j) qs(t, left, j);
    if (i<right) qs(t, i, right);
}
```

Searching

- Mencari data berdasarkan key tertentu.
- Kelompok Algoritme:
  - Sequential search
  - Sorted array search
  - Hashing
  - Recursive structures search
  - Multidimensional search
- Yang dibahas pada matakuliah ini adalah sequential search dan sorted array search.

Sequential search

- Setiap elemen dalam array ditelusuri satu per satu dari arah kiri hingga ketemu atau data yang dibaca telah habis.
- Mudah diimplementasikan.
- Waktu proses relatif lambat untuk ukuran data sangat besar.

Sequential search

```
// Output : indeks letak data pertama ditemukan
// atau bernilai -1 jika tidak ditemukan

int sequentialSearch(LIST t[], int n, int val)
{
    int i;

    for(i=0; i<n; i++)
       if(val==t[i].key) return i;

    return -1; /* no match */
}
```


Sequential sorted array search

- Data harus terurut
- Setiap elemen dalam array ditelusuri satu per satu dari arah kiri hingga ketemu atau key data yang dibaca lebih besar dari nilai yang dicari atau data telah habis.
- Mudah diimplementasikan.
- Waktu proses relatif cepat untuk kasus data yang dicari terletak di bagian depan.

Sequential sorted array search

```
// Output : indeks letak data pertama ditemukan
// atau bernilai -1 jika tidak ditemukan

int sortedSearch(LIST t[], int n, int val)
{
    int i=0, hsl=-1;

    while ((t[i].key<=val) && (i<n) && (hsl==-1)) {
       if(t[i].key==val) hsl=i;
       i++;
    }
    return hsl; /* no match */
}
```


Binary search

- Data harus terurut.
- Langsung dilihat data yang terletak di tengah (misalnya data ke-*mid*).
- Jika key data ke-*mid* lebih besar dari yang dicari, maka pencarian diarahkan ke daerah sebelah kiri dari *mid*. Sebaliknya, diarahkan ke sebelah kanan.

Binary search

```
int binarySearch(LIST t[], int n, int val)
{
    int left, right, mid;

    left = 0; right = n-1;
    while(left<=right) {
        mid=(left+right)/2;
        if (val<t[mid].key)
           right=mid-1;
        else if (val>t[mid].key)
           left=mid+1;
        else
           return mid; /* found */
    }
    return -1; /* not found */
}
```