Algoritme dan Pemrograman

• *Sorting* (Pengurutan)

Sorting (Pengurutan)

- Mengurutkan data berdasarkan kunci tertentu.
- Jenis *sorting*:
  - *Ascending* (menaik)
  - *Descending* (menurun)
- Manfaat: mempercepat dan memudahkan akses data
- Banyak algoritme *sorting* dikembangkan untuk:
  - Mempercepat proses
  - Mengefisienkan penggunaan sumberdaya
- Ingat syarat terurut pada *binary search*

Contoh

- Masukan:

  5 2 4 6 1 3

- Keluaran:

  1 2 3 4 5 6
  (*ascending*)

  6 5 4 3 2 1
  (*descending*)

Data untuk *sorting*

- Anggaplah kita mempunyai suatu *array* integer
- Akan dilakukan *sorting* secara *ascending*
- *Sorting* secara *descending* dapat dilakukan dengan penyesuaian pada algoritme

Fungsi pendukung

- Mempertukarkan nilai t1 dan t2:

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

- Menampilkan n buah data

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


Algoritme #1: *BUBBLE SORT*

- Ide: nilai yang besar seperti "gelembung" yang akan "naik"
- Keuntungan:
  - Sederhana dan mudah diimplementasikan
  - Cepat jika data masukan sudah terurut
- Kelemahan:
  - Secara umum membutuhkan waktu proses lebih lama
  - Biasa dijadikan contoh algoritme pengurutan yang tidak efisien ($O(n^2)$)

Cara kerja

- 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.
- Pada putaran ke-i, nilai ke-i terbesar akan berada di posisi yang benar
  - Putaran pertama akan menempatkan nilai paling besar di akhir *array*
  - Dimanfaatkan untuk optimasi *inner loop* agar tidak lagi memeriksa nilai yang telah benar posisinya

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]);
        printf("%d %d |", i, j);
        printlist(x, n);
    }
  }
}
```

Bagian yang **ditebalkan** adalah optimasi *inner loop*

Bagian berwarna biru hanya untuk *tracing*

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

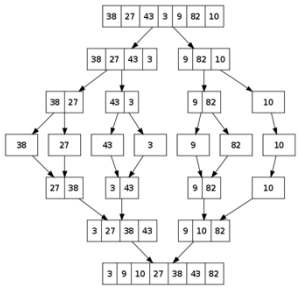
- Baca elemen dalam array dari kiri ke kanan, dan tempatkan pada posisi yang sesuai berdasarkan nilai key.
- Keuntungan:
  - Sederhana dan mudah diimplementasikan
- Kelemahan:
  - Kurang efisien untuk data berukuran besar

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

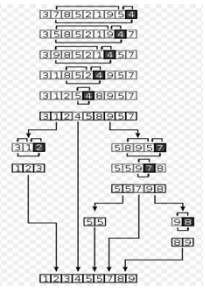
```
void mergeSort(LIST t[], int n)
{
   ms(t, 0, n-1);
}

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

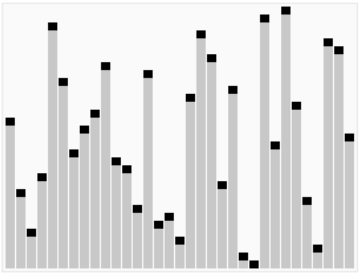
```
merge(LIST t[], int left, int right, int mid) {
   LIST c[SIZE];
   int i=left, j=mid+1, k=left;
   while ((i<=mid)&&(j<=right)) {
      if(t[i].key<t[j].key) {
         c[k]=t[i]; k++; i++;
      } else {
         c[k]=t[j]; k++; j++;
      }
   }
   while (i<=mid) {
      c[k]=t[i]; k++; i++;
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

DEPARTEMEN ILMU KOMPUTER
INSTITUT PERTANIAN BOGOR