Laporan Praktikum 4  
Praktikum Analisis Algoritma



Imron Madani (140810170061)

Fakultas Matematika dan Ilmu Pengetahuan Alam  
Program Studi Teknik Informatika  
Universitas Padjajaran  
2019

# Studi Kasus

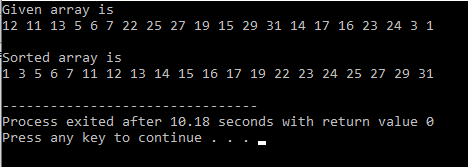
## **Studi Kasus 1: MERGE SORT**

Setelah Anda mengetahui Algoritma Merge-Sort mengadopsi paradigma divide & conquer, lakukan Hal berikut:

1. Buat program Merge-Sort dengan bahasa C++

|  |
| --- |
| /\* C program for Merge Sort \*/  #include<stdlib.h>  #include<stdio.h>  // Merges two subarrays of arr[].  // First subarray is arr[l..m]  // Second subarray is arr[m+1..r]  void merge(int arr[], int l, int m, int r)  {  int i, j, k;  int n1 = m - l + 1;  int n2 = r - m;  /\* create temp arrays \*/  int L[n1], R[n2];  /\* Copy data to temp arrays L[] and R[] \*/  for (i = 0; i < n1; i++)  L[i] = arr[l + i];  for (j = 0; j < n2; j++)  R[j] = arr[m + 1+ j];  /\* Merge the temp arrays back into arr[l..r]\*/  i = 0; // Initial index of first subarray  j = 0; // Initial index of second subarray  k = l; // Initial index of merged subarray  while (i < n1 && j < n2)  {  if (L[i] <= R[j])  {  arr[k] = L[i];  i++;  }  else  {  arr[k] = R[j];  j++;  }  k++;  }  /\* Copy the remaining elements of L[], if there  are any \*/  while (i < n1)  {  arr[k] = L[i];  i++;  k++;  }  /\* Copy the remaining elements of R[], if there  are any \*/  while (j < n2)  {  arr[k] = R[j];  j++;  k++;  }  }  /\* l is for left index and r is right index of the  sub-array of arr to be sorted \*/  void mergeSort(int arr[], int l, int r)  {  if (l < r)  {  // Same as (l+r)/2, but avoids overflow for  // large l and h  int m = l+(r-l)/2;  // Sort first and second halves  mergeSort(arr, l, m);  mergeSort(arr, m+1, r);  merge(arr, l, m, r);  }  }  /\* UTILITY FUNCTIONS \*/  /\* Function to print an array \*/  void printArray(int A[], int size)  {  int i;  for (i=0; i < size; i++)  printf("%d ", A[i]);  printf("\n");  }  /\* Driver program to test above functions \*/  int main()  {  int arr[] = {12, 11, 13, 5, 6, 7, 22, 25, 27, 19, 15, 29, 31, 14, 17, 16, 23, 24, 3, 1};  int arr\_size = sizeof(arr)/sizeof(arr[0]);  printf("Given array is \n");  printArray(arr, arr\_size);  mergeSort(arr, 0, arr\_size - 1);  printf("\nSorted array is \n");  printArray(arr, arr\_size);  return 0;  } |

1. Kompleksitas waktu algoritma merge sort adalah O(n lg n). Cari tahu kecepatan komputer Anda dalam memproses program. Hitung berapa running time yang dibutuhkan apabila input untuk merge sort-nya adalah 20?



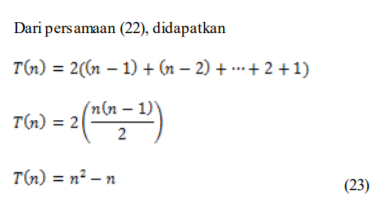
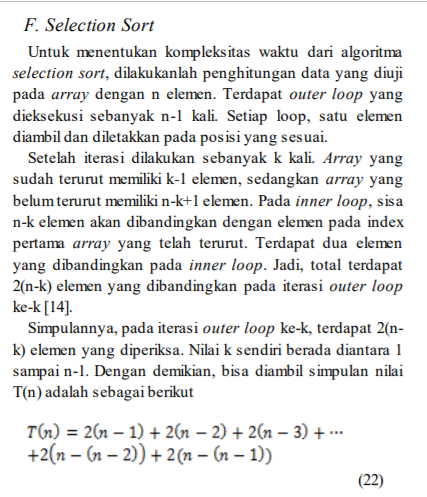
## **Studi Kasus 2: SELECTION SORT**

Selection sort merupakan salah satu algoritma sorting yang berparadigma divide & conquer. Untuk membedah algoritma selection sort, lakukan langkah-langkah berikut:

* Pelajari cara kerja algoritma selection sort

|  |  |  |
| --- | --- | --- |
| Dikatakan *selection sort* karena algoritma ini mencoba **memilih** satu per satu elemen data dari posisi awal, untuk mencari data paling kecil dengan mencatat posisi index-nya saja, lalu dilakukan **pertukaran hanya sekali** pada akhir setiap tahapan.  Algoritma Selection Sort dilakukan untuk **menyempurnakan kekurangan** dari *bubble sort* yang melakukan pertukaran setiap kali perbandingan memenuhi kriterianya.   |  | | --- | | <http://4.bp.blogspot.com/_oeolvjNvF3s/TQQoP-xj3MI/AAAAAAAAAIM/Dq86RbAQr7A/s1600/Selection.GIF> | | Gambar: Hasil *running* |   Algoritma dari *selection sort* adalah sebagai berikut:   1. inisialisasi n adalah ukuran data, pada contoh di atas n = 7 2. inisialisasi i ← 1 sebagai awal proses. 3. kecil ← i  { asumsi awal, posisi ke-i lah yang paling kecil, **garis merah** pada gambar di atas }. 4. j ← i + 1  { j adalah posisi data pembanding, **garis hitam** pada gambar di atas } 5. Jika Data j < Data kecil, maka posisi ke-j lebih kecil, lalu ubah posisi index, kecil  ← j 6. j ← j + 1 7. Jika j = n { perbandingan sudah sampai index terakhir } maka lanjut ke baris 8, jika tidak ulang ke baris 5. 8. lakukan pertukaran Data-i  dengan Data-terkecil (**garis merah** dengan **garis biru** pada gambar di atas). 9. i  ← i + 1 { perubahan tiap tahap, **garis merah** bergeser pada tiap tahapan } 10. Ulang ke baris ke-3 selama i < n - 1 11. Selesai . |

* Tentukan T (n) dari rekurensi (pengulangan) selection sort berdasarkan penentuan rekurensi divide & conquer:



* Selesaikan persamaan rekurensi T(n) dengan **metode recursion-tree** untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big-Ω, dan Big-Θ

Big-O, Big-Ω, dan Big-Θ masing-masing adalah n2

* Lakukan implementasi koding program untuk algoritma selection sort dengan menggunakan bahasa C++

|  |
| --- |
| // C++ program for implementation of selection sort  #include <bits/stdc++.h>  using namespace std;  void swap(int \*xp, int \*yp)  {  int temp = \*xp;  \*xp = \*yp;  \*yp = temp;  }  void selectionSort(int arr[], int n)  {  int i, j, min\_idx;  // One by one move boundary of unsorted subarray  for (i = 0; i < n-1; i++)  {  // Find the minimum element in unsorted array  min\_idx = i;  for (j = i+1; j < n; j++)  if (arr[j] < arr[min\_idx])  min\_idx = j;  // Swap the found minimum element with the first element  swap(&arr[min\_idx], &arr[i]);  }  }  /\* Function to print an array \*/  void printArray(int arr[], int size)  {  int i;  for (i=0; i < size; i++)  cout << arr[i] << " ";  cout << endl;  }  // Driver program to test above functions  int main()  {  int arr[] = {64, 25, 12, 22, 11};  int n = sizeof(arr)/sizeof(arr[0]);  selectionSort(arr, n);  cout << "Sorted array: \n";  printArray(arr, n);  return 0;  }  // This is code is contributed by rathbhupendra |

### **Studi Kasus 3: INSERTION SORT**

Insertion sort merupakan salah satu algoritma sorting yang berparadigma divide & conquer. Untuk membedah algoritma insertion sort, lakukan langkah-langkah berikut:

* Pelajari cara kerja algoritma insertion sort

Cara kerja insertion sort sebagaimana namanya. Pertama-tama, dilakukan iterasi, dimana di setiap iterasi insertion sort memindahkan nilai elemen, kemudian menyisipkannya berulang-ulang sampai ketempat yang tepat. Begitu seterusnya dilakukan. Dari proses iterasi, seperti biasa, terbentuklah bagian yang telah di-sorting dan bagian yang belum di-sorting.

Metode insertion sort ini bekerja dengan cara yang serupa dengan apa yang biasa dilakukan oleh seorang pemain kartu professional, yaitu menyisipkan kartu pada suatu lokasi tertentu di bagian yang telah terurut (kita anggap di bagian sebelah kiri). Jika kita menganggap kartu yang berada di tangan (kita anggap di bagian sebelah kiri) merupakan bagian larik yang sudah terurut, dan menganggap “tabel” di sebelah kanan merupakan bagian yang belum terurut, kita akan dapa mengembangkan teknik penguruatan yang dinamakan insertion sort dengan mengambil kartu tertentu dari sebelah kanan kemudian menyisipkannya ke tempat yang sesuai di sebelah kiri.

Algoritma Insertion Sort dapat dirangkum sebagai berikut:

Simpan nilai Ti kedalam variabel sementara, dengan i = 1.

Bandingkan nilainya dengan elemen sebelumnya.

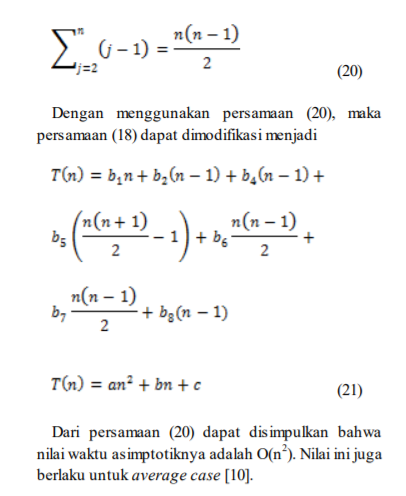
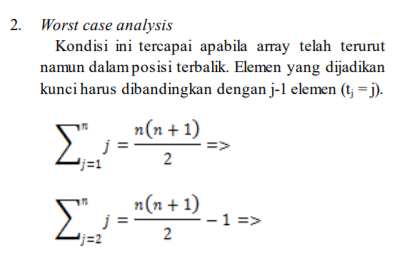
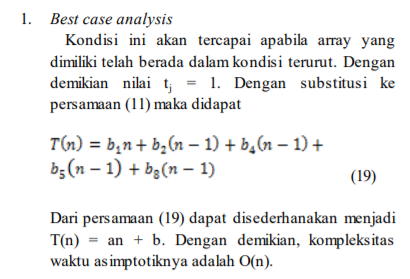
Jika elemen sebelumnya (Ti-1) lebih besar nilainya daripada Ti, maka tindih nilai Ti dengan nilai Ti-1 tersebut. Decrement i (kurangi nilainya dengan 1).

Lakukan terus poin ke-tiga, sampai Ti-1 ≤ Ti.

Jika Ti-1 ≤ Ti terpenuhi, tindih nilai di Ti dengan variabel sementara yang disimpan sebelumnya.

Ulangi langkah dari poin 1 di atas dengan i di-increment (ditambah satu).

* Tentukan T(n) dari rekurensi (pengulangan) insertion sort berdasarkan penentuan rekurensi divide & conquer:



* Selesaikan persamaan rekurensi T(n) dengan **metode** **subtitusi** untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big-Ω, dan Big-Θ

Big-O = n

Big-Ω = n2

Big-Θ n2

* Lakukan implementasi koding program untuk algoritma insertion sort dengan menggunakan bahasa C++

|  |
| --- |
| // C++ program for insertion sort  #include <bits/stdc++.h>  using namespace std;  /\* Function to sort an array using insertion sort\*/  void insertionSort(int arr[], int n)  {  int i, key, j;  for (i = 1; i < n; i++)  {  key = arr[i];  j = i - 1;  /\* Move elements of arr[0..i-1], that are  greater than key, to one position ahead  of their current position \*/  while (j >= 0 && arr[j] > key)  {  arr[j + 1] = arr[j];  j = j - 1;  }  arr[j + 1] = key;  }  }  // A utility function to print an array of size n  void printArray(int arr[], int n)  {  int i;  for (i = 0; i < n; i++)  cout << arr[i] << " ";  cout << endl;  }  /\* Driver code \*/  int main()  {  int arr[] = { 12, 11, 13, 5, 6 };  int n = sizeof(arr) / sizeof(arr[0]);  insertionSort(arr, n);  printArray(arr, n);  return 0;  }  // This is code is contributed by rathbhupendra |

### **Studi Kasus 4: BUBBLE SORT**

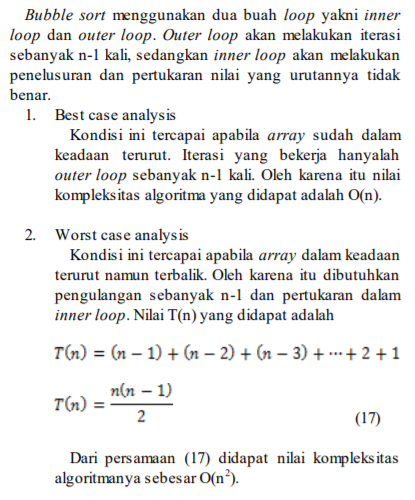
Bubble sort merupakan salah satu algoritma sorting yang berparadigma divide & conquer. Untuk membedah algoritma bubble sort, lakukan langkah-langkah berikut:

* Pelajari cara kerja algoritma bubble sort

Algoritma Bubble Sort ini merupakan proses pengurutan yang secara berangsur-angsur berpindah ke posisi yang tepat karena itulah dinamakan Bubble yang artinya gelembung. Algoritma ini akan mengurutkan data dari yang terbesar ke yang terkecil (ascending) atau sebaliknya (descending).

Secara sederhana, bisa didefenisikan algoritma Bubble Sort adalah pengurutan dengan cara pertukaran data dengan data disebelahnya secara terus menerus sampai dalam satu iterasi tertentu tidak ada lagi perubahan.

* Tentukan T(n) dari rekurensi (pengulangan) insertion sort berdasarkan penentuan rekurensi divide & conquer:



* Selesaikan persamaan rekurensi T(n) dengan **metode** **master** untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big-Ω, dan Big-Θ

Big-O = n

Big-Ω = n2

Big-Θ n2

* Lakukan implementasi koding program untuk algoritma bubble sort dengan menggunakan bahasa C++

|  |
| --- |
| // C program for implementation of Bubble sort  #include <stdio.h>  void swap(int \*xp, int \*yp)  {  int temp = \*xp;  \*xp = \*yp;  \*yp = temp;  }  // A function to implement bubble sort  void bubbleSort(int arr[], int n)  {  int i, j;  for (i = 0; i < n-1; i++)  // Last i elements are already in place  for (j = 0; j < n-i-1; j++)  if (arr[j] > arr[j+1])  swap(&arr[j], &arr[j+1]);  }  /\* Function to print an array \*/  void printArray(int arr[], int size)  {  int i;  for (i=0; i < size; i++)  printf("%d ", arr[i]);  printf("n");  }  // Driver program to test above functions  int main()  {  int arr[] = {64, 34, 25, 12, 22, 11, 90};  int n = sizeof(arr)/sizeof(arr[0]);  bubbleSort(arr, n);  printf("Sorted array: \n");  printArray(arr, n);  return 0;  } |