**Analyze Worst-Case & Average-Case**

**Run Time of Quick Sort**

**Worst-case**

The answer depends on strategy for choosing pivot. In early versions of Quick Sort where leftmost (or rightmost) element is chosen as pivot, the worst occurs in following cases.

1) Array is already sorted in same order.

Assume the input of the Quicksort is a sorted array and we choose the leftmost/rightmost element as a pivot element. In this case, we’ll have two extremely u

nbalanced arrays. One array will have one element and the other one will have elements.

2) Array is already sorted in reverse order.

When the given input array is sorted reversely and we choose the leftmost/rightmost element as the pivot element, the worst case occurs. Again, in this case, the pivot elements will split the input array into two unbalanced arrays.

3) All elements are same (special case of case 1 and 2)

**Average-case**

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are [uniformly distributed](http://en.wikipedia.org/wiki/Uniform_distribution_%28discrete%29) (including the case of x not being present in array). So we sum all the cases and divide the sum by (n+1). Following is the value of average case time complexity.

**Analyze Worst-Case & Average-Case**

**Run Time of Quick Sort**

**Worst-case**

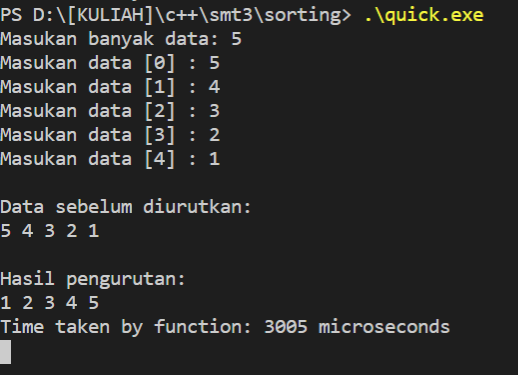
Bergantung pada strategi pemilihan pivot. Pada versi awal dari Quick Sort dimana elemen paling kiri atau paling kanan dipilih sebagai pivot, worst case terjadi dalam kasus sebagai berikut :

1) Array sudah terurut dari kecil ke besar.

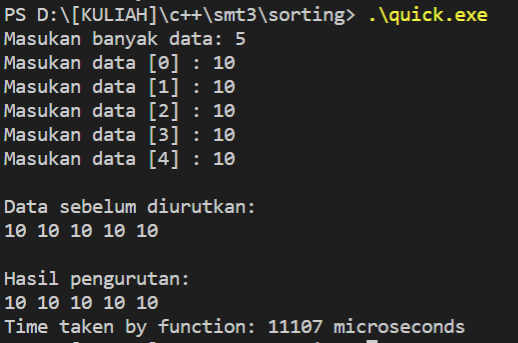
Diasumsikan bahwa input pada Quick sort adalah array yang sudah terurut dari kecil ke besar dan kita memilih elemen paling kanan atau paling kiri sebagai pivot. Pada kasus ini, kita akan memiliki 2 array yang sangat tidak seimbang. Satu array akan memiliki hanya 1 elemen sedang array yang lain akan memiliki (n-1) elemen.

2) Array sudah terurut dari besar ke kecil.

Ketika array yang diinputkan sudah terurut dari besar ke kecil dan kita memilih elemen paling kiri atau elemen paling kanan sebagai pivot, maka akan terjadi worst case, dimana elemen akan terbagi kedalam dua array yang tidak seimbang.



3) Semua elemen memiliki nilai yang sama.



**Average-case**

Dalam analisis average case, kita mengambil semua kemungkinan input dan menghitung waktu komputasi untuk semua input. Jumlahkan semua nilai yang terhitung dan bagi dengan jumlah bilangan input. Kita harus mengetahui (atau memprediksi) distribusi kasus. Untuk setiap kasus linear search, asumsikan bahwa semua kasus terdistribusi secara merata (termasuk kasus x tidak ada dalam array). Sehingga kita dapat menjumlahkan semua kasus dan membagi hasil penjumlahan dengan (n+1). Menghasilkan nilai rata-rata kompleksitas waktu.

Source Code dalam pengerjaan soal

|  |
| --- |
| #include <iostream>  #include <conio.h>  #include <algorithm>  #include <chrono>  using namespace std;  using namespace std::chrono;  void quick\_sort(int arr[], int left, int right)  {      int i = left, j = right;      int tmp;      int pivot = arr[(left + right) / 2]; /\* partition \*/      while (i < j)      {          while (arr[i] < pivot)              i++;          while (arr[j] > pivot)              j--;          if (i <= j)          {              tmp = arr[i];              arr[i] = arr[j];              arr[j] = tmp;              i++;              j--;          };      }; /\* recursion \*/      if (left < j)          quick\_sort(arr, left, j);      if (i < right)          quick\_sort(arr, i, right);  }  int main()  {      int i, n, data[50];      cout << "Masukan banyak data: ";      cin >> n;      for (i = 0; i < n; i++)      {          cout << "Masukan data [" << i << "] : ";          cin >> data[i];      }      cout << "\nData sebelum diurutkan: " << endl;      for (i = 0; i < n; i++)      {          cout << data[i] << " ";      }      cout << "\n";      auto start = high\_resolution\_clock::now();      quick\_sort(data, 0, n - 1); //hasil pengurutan      cout << "\nHasil pengurutan:\n";      {          int i;          for (i = 0; i < n; i++)              cout << data[i] << " ";          cout << "\n";      }      auto stop = high\_resolution\_clock::now();        auto duration = duration\_cast<microseconds>(stop - start);      cout << "Time taken by function: "           << duration.count() << " microseconds" << endl;      getch();  } |