

BLG335E Analysis of Algorithm

Project 1



21 / 10 / 2016

Istanbul TechnIcal UniversIty

Name : Emre ÖZDİL

Number : 150120138

Date : 21.10.2016

Teacher : Hazım Kemal Ekenel

CRN : 11924

# **A - Asymptotic Upper Bound**

Merge Sort is run on O(nlg(n)).

My code for Merge Sort:

if (lowerBound < upperBound)

{

int median = (lowerBound + upperBound) / 2;

sort(unsorted, lowerBound, median); O(1)

sort(unsorted, median + 1, upperBound);

merge(unsorted, lowerBound, median, upperBound);

}

T(n) = 2T(n/2) + merge part

Merge Part:

for (int i = 0; i < leftLength; i++)

{

Left[i] = unsorted[lowerBound + i];

}

for (int j = 0; j < rightLength; j++)

{

Right[j] = unsorted[median + 1 + j];

}

for (int k = lowerBound; k <= upperBound; k++) run (upperBound-lowerBound)

{ times maximum so it depens n

if (Left[i].distance <= Right[j].distance) and runs on O(n)

{

unsorted[k] = Left[i];

i++;

}

else

{

unsorted[k] = Right[j];

j++;

}

}

These fors run on O(n). So it is T(n) = 2T(n/2) + O(n). It becomes finally 1 due to binary division so the result is O(n) + O(n) + … + O(n) There are lg(n) times O(n). It shows that it is O(nlg(n)).

Insertion Sort is run on O(n2).

My code for Insertion Sort:

int j;

Point temp;

for (int i = 1; i < length; i++)

{

j = i;

while (j > 0 && unsorted[j].distance < unsorted[j - 1].distance for loop n times

{ while loop n times;

temp = unsorted[j]; therefore, O(n2)

unsorted[j] = unsorted[j - 1];

unsorted[j - 1] = temp;

j--;

}

}

It shows that it is O(n2).

Linear Search is run on O(n2).

My code for Linear Search:

while (j != total)

{

for (int i = 0; i < length; i++)

{

if (unsorted[i].distance > higherDistance)

{

higherDistance = unsorted[i].distance;

higherIndex = i;

}

}

for (j ; j < total; j++) (n-k)\*k complexity

{ Best case O(1)

if (unsorted[j].distance < higherDistance) Worst case O(n2/4)=O(n2)

{

higherDistance = unsorted[j].distance;

temp = unsorted[j];

unsorted[j] = unsorted[higherIndex];

unsorted[higherIndex] = temp;

break;

}

}

}

Worst case => N = 1000000, K = 500000

# **B – Calculation Times**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **K** | **Merge Sort** | **Insertion Sort** | **Linear Search** |
| 10 | 1 | 0 | 0 | 0 |
| 10 | 2 | 0 | 0 | 0 |
| 10 | 5 | 0 | 0 | 0 |
| 100 | 1 | 0 | 0 | 0 |
| 100 | 2 | 0 | 0 | 0 |
| 100 | 10 | 0 | 0 | 0 |
| 100 | 50 | 0 | 0 | 0 |
| 1k | 1 | 0 | 0.002 | 0 |
| 1k | 2 | 0 | 0.002 | 0 |
| 1k | 10 | 0 | 0.002 | 0 |
| 1k | 500 | 0 | 0.002 | 0 |
| 10k | 1 | 0.005 | 0.208 | 0 |
| 10k | 2 | 0.005 | 0.208 | 0 |
| 10k | 10 | 0.005 | 0.208 | 0 |
| 10k | 5k | 0.005 | 0.208 | 0.047 |
| 100k | 1 | 0.049 | 20.851 | 0 |
| 100k | 2 | 0.049 | 20.851 | 0 |
| 100k | 10 | 0.049 | 20.851 | 0 |
| 100k | 50k | 0.049 | 20.851 | 4.696 |
| 1m | 1 | 0.511 | 2164.27 | 0.003 |
| 1m | 2 | 0.511 | 2164.27 | 0.003 |
| 1m | 10 | 0.511 | 2164.27 | 0.003 |
| 1m | 500k | 0.511 | 2164.27 | 492.604 |

\*The unit is second

# **C – Graph**

It is clearly seen that Merge Sort is better in these algorithms. Merge Sort is O(nlg(n)) because it has lower asymtotic function view on the graph.

Until N =10000 all algorithms almost have same time. K factor affects only Linear Search algorithm because Linear Search algorithm complexity depends on K factor.

If (n-k)\*k is small number I select Linear Search Algorithm. Otherwise Merge Sort is better than Linear Search. Insertion Sort is worst algorithm in these algoritgms.