**Analysis of Algorithms Homework 1 Report**

**Talha Çomak, 150160726**

**a)**

for Merge algorithm:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Statement | Steps | frequency | | Total steps | |
| If true | If false | If true | If false |
| 1 | x = m - l + 1; int y = h-m; | 1 | 1 | 1 | 1 | 1 |
| 2 | for (i = 0; i < x; i++) | 1 | n/2+1 | n/2+1 | n/2+1 | n/2+1 |
| 3 | L.push\_back(vec[l + i]); | 1 | n/2 | n/2 | n/2 | n/2 |
| 4 | for (j = 0; j < y; j++) | 1 | n/2+1 | n/2+1 | n/2+1 | n/2+1 |
| 5 | R.push\_back(vec[m + 1 + j]); | 1 | n/2 | n/2 | n/2 | n/2 |
| 6 | i = 0; j = 0; k = l; | 3 | 3 | 3 | 3 | 3 |
| 7 | while (i < x && j < y) | 2 | n/2 | n/2 | n | n |
| 8 | if (L[i] <= R[j]) | 1 | n/2-1 | n/2-1 | n/2-1 | n/2-1 |
| 9 | vec[k] = L[i]; i++; | 2 | n/2-1 (if) | 0 | n-2 | 0 |
| 10 | vec[k] = R[j]; j++; | 2 | 0 (else) | n/2-1 | 0 | n-2 |
| 11 | k++; | 1 | n/2-1 | n/2-1 | n/2-1 | n/2-1 |
| 12 | while (i < x) | 1 | 1 | 1 | 1 | 1 |
| 13 | vec[k] = L[i]; i++; k++; | 3 | 1 | 0 | 3 | 0 |
| 14 | while (j < y) | 1 | 1 | 1 | 1 | 1 |
| 15 | vec[k] = R[j]; j++; k++; | 3 | 1 | 0 | 3 | 0 |
| T | Total |  |  |  | 9n/2+10 | 9n/2+6 |

Worst case is 9n/2+10 so worst time complexity = O(n) [linear] => asymtotic upper bound

Best case is 9n/2+10 so best time complexity = Ω(n) [linear] => asymtotic lower bound

For Bubble Sort:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Statement | Steps | frequency | | Total steps | |
| If true | If false | If true | If false |
| 1 | bool sorted = false; | 1 | 1 | 1 | 1 | 1 |
| 2 | for (int i = n; i > 1 && sorted == false; i--) | 2 | n | n | 2n | 2n |
| 3 | sorted = true; | 1 | n-1 | 1 | n-1 | 1 |
| 4 | for (int j = 1; j < i; j++) | 1 | n(n+1)/2 | n | n(n+1)/2 | n |
| 5 | if (vec[j] < vec[j - 1]) | 1 | (n-1)n/2 | n-1 | (n-1)n/2 | n-1 |
| 6 | k = vec[j];  vec[j] = vec[j - 1];  vec[j - 1] = k;  sorted = false; | 4 | (n-1)n/2 | 0 | 4(n-1)n/2 | 0 |
| T | Total |  |  |  | 3n^2 +3n | 4n+1 |

Worst case is 3n^2 +3n so worst time complexity = O(n^2) [quadratic] => asymtotic upper bound

Best case is n^2 +3n so best time complexity = Ω(n) [linear] => asymtotic lower bound

**b)**

Bubble sort for unsorted:

n=1000 -> 0.084 sn,n=10000 -> 8.585 sn, n=100000 -> 710 sn, n=1.000.000 ->approximately 70000 sn

Bubble sort for sorted:

n=1000 -> 0 sn, n=10000 -> 0.001 sn, n=100000 -> 0.012 sn, n=1.000.000 -> 0.066 sn

Merge for unsorted:

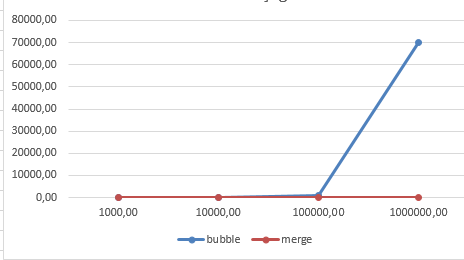
n=1000 -> 0.029 sn, n=10000 -> 0.255 sn, n=100000 -> 2.397 sn, n=1.000.000 -> 22.421

Merge for sorted:

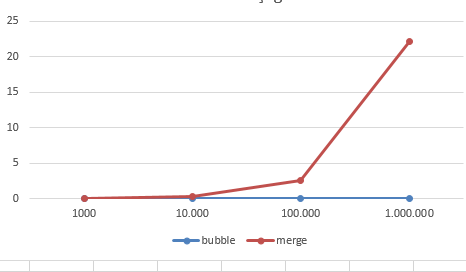
n=1000 -> 0.020 sn, n=10000 -> 0.249 sn, n=100000 -> 2.541 sn, n=1.000.000 -> 22.198

**c)**

for unsorted:



for sorted:



In a result for aprroximately sorted values bubble sort is more efficient than merge, but for more messy cases merge is more efficient than bubble by far. But if we don’t know about mess of data; when the data is low we should use merge and when the data is large we certainly should use merge.

**d)**

mystery:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Statement | Steps | frequency | | Total steps | |
| If true | If false | If true | If false |
| 1 | Algorithm Mystery(n) | 0 | 0 | 0 | 0 | 0 |
| 2 | r <- 0 | 1 | 1 | 1 | 1 | 1 |
| 3 | for i <- 1 to n do | 1 | n+1 | n+1 | n+1 | n+1 |
| 4 | for j <- i+1 to n do | 1 | n(n+1)/2 | n(n+1)/2 | n(n+1)/2 | n(n+1)/2 |
| 5 | for k <- 1 to j do | 1 | (2n^3+3n^2-7n)/6 | (2n^3+3n^2-7n)/6 | (2n^3+3n^2-7n)/6 | (2n^3+3n^2-7n)/6 |
| 6 | r <- r+1; | 1 | (2n^3+3n^2-4n+6)/6 | (2n^3+3n^2-4n+6)/6 | (2n^3+3n^2-4n+6)/6 | (2n^3+3n^2-4n+6)/6 |
| 7 | return r | 0 | 0 | 0 | 0 | 0 |
| T |  |  |  |  | (2n^3+6n^2+5n+18)/6 | (2n^3+6n^2+5n+18)/6 |

Worst case is (2n^3+6n^2+2n+6)/6 so worst time complexity = O(n^3) [cubic] => asymtotic upper bound

Best case is (2n^3+6n^2+2n+6)/6 so best time complexity = Ω(n^3) [cubic] => asymtotic lower bound