SECOND ASSIGNMENT ADS

Solved by :

Shorouk G. AWWAD

30002030

22.2.2019

```
#include <iostream>#include<math.h>    #include <string> #include <cstdlib> #include <ctime> #include <chrono>
  #include <limits> #include <chrono> #include <algorithm>
using namespace std;
void mergesort(float * A, int p, int r);
void merge2(float * A, int p, int q, int r);
void mergsort2(float * A, int p, int r, int c);
void insertion(float * A, int p, int r);
int main() {
      int n;
      srand((unsigned) time(0));
      cout << "enter the number of elements:";
      cin >> n;
//
      float A[n];
      for (int i = 0; i < n; i++) {
            cin >> A[i];
 }
      mergsort2(A, 0, n - 1, c);
      for (int i = 0; i < n; i++) {
            cout << i << ": " << A[i] << endl;
      }
}
//insertion sort and merge sort
void mergsort2(float * A, int p, int r, int c) {
      if (r - p + 1 <= c) {
            insertion(A, p, r);

      } else {
 int q;
   q = (p + r) / 2;
            mergsort2(A, p, q, c);
            mergsort2(A, q + 1, r, c);
            merge2(A, p, q, r);
      }
      //insertion sort
}
```

```
void insertion(float * A, int p, int r) {
  float Key;
   int j;
  for (int i = p + 1; i <= r; i++) {
              Key = A[i];
              j = i - 1;
     while (j >= p && A[j] > Key) {
                    A[j + 1] = A[j];
                    j--;       }
        A[j + 1] = Key;
        }
}
void merge2(float * 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++;
        }
```

Q2.1

b)

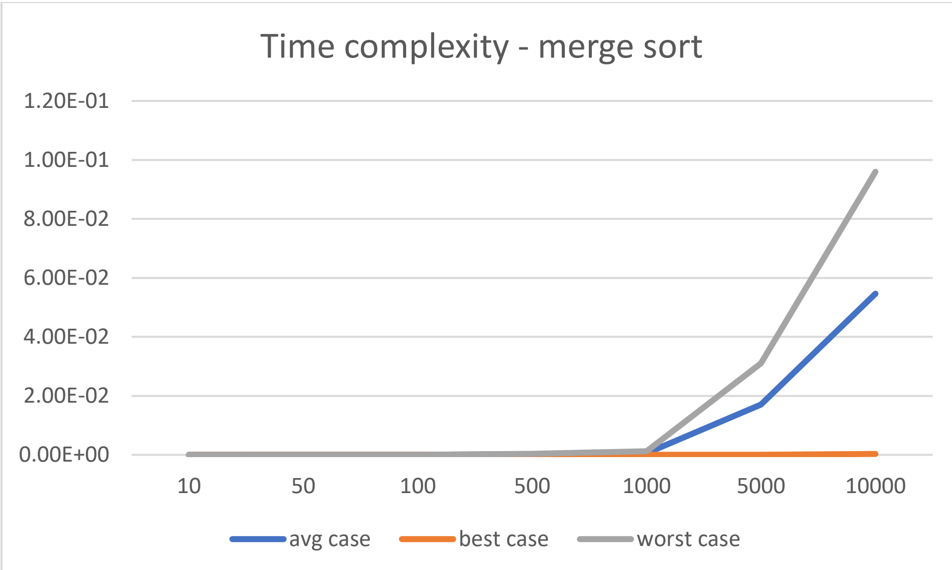
Selection sort Algorithm:

n: number of elements	n: number of elements2	n: number of elements3	n: number of elements4
10	8.31E-07	7.44E-07	7.47E-07
100	2.48E-05	2.06E-05	2.70E-05
500	5.00E-04	4.60E-04	6.40E-04
1000	3.25E-03	3.17E-03	3.83E-03
5000	4.75E-02	3.01E-02	5.67E-02
10000	1.36E-01	1.13E-01	1.58E-01

Merge/ insertion sort results :

k	n	avg case	best case	worst case
5	10	6.22E-07	3.34E-07	4.58E-07
25	50	4.59E-06	1.35E-06	5.85E-06
50	100	1.33E-05	2.49E-06	2.00E-05
5000	10000	5.46E-02	3.13E-04	9.60E-02

In those measurements k has been varied to be always half the number of the entries. For example, if n=10000, k= 5000 (the last case)

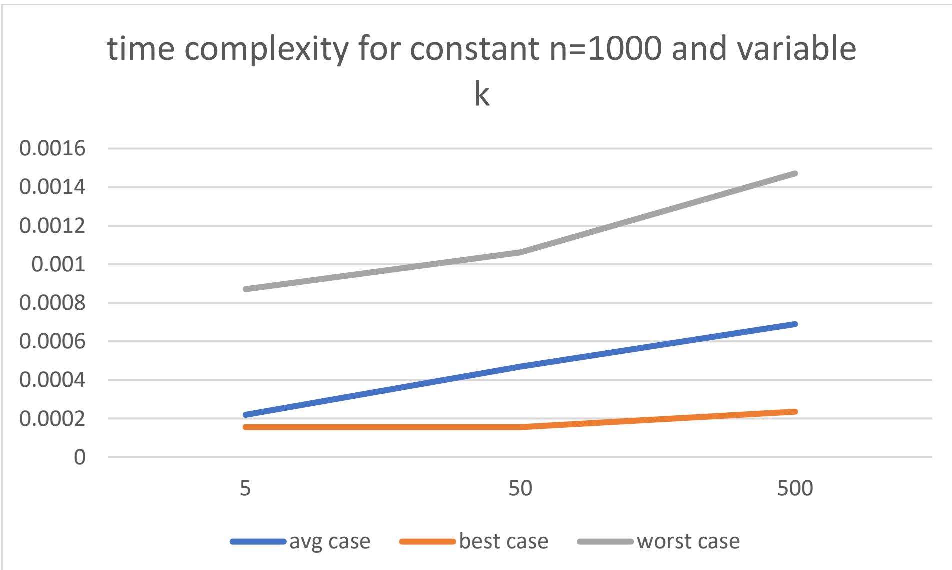


k	n	avg case	best case	worst case
5	10	6.22E-07	3.34E-07	4.58E-07
25	50	4.59E-06	1.35E-06	5.85E-06
50	100	1.33E-05	2.49E-06	2.00E-05
250	500	1.91E-04	8.94E-06	3.62E-04
500	1000	6.10E-04	1.42E-05	1.20E-03
2500	5000	1.70E-02	6.55E-05	3.10E-02
5000	10000	5.46E-02	3.13E-04	9.60E-02

Comparing the results from both Algorithms to the same corresponding number of elements, using the merge insertion sort is more way less expensive in the term of time complexity, which means for the same number of entries, the algorithm (Merge) takes far less time to execute, which makes it more effective in most of cases, especially in the worst and the best case.

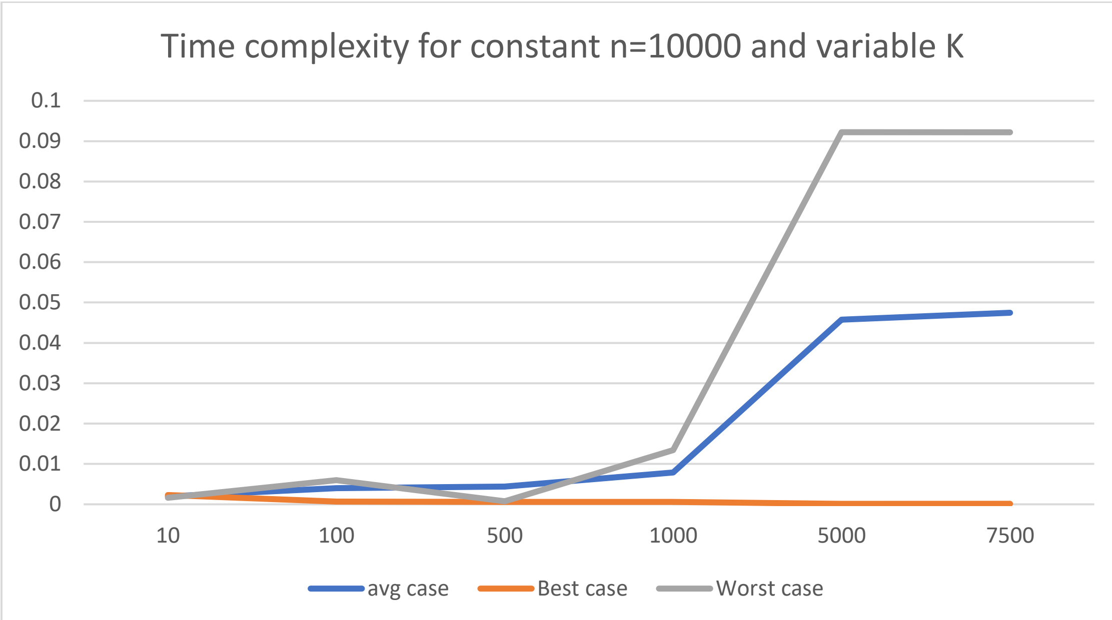
Q2.1

c)



For increasing k, the insertion sort increases compared to merge sort which means the time complexity becomes higher and slower. And for small k, the algorithm becomes faster. Even if this seems to be definite, it is not. For some K at the middle -in the above table: near to K=50 -- there is a k at which the time complexity graph forms a minimum.

n	k	avg case	Best case	Worst case
10000	10	0.0019603	0.00229512	0.0016022
10000	100	0.004	0.0007	0.006
10000	500	0.004367	0.0006166	0.0007495
10000	1000	0.0079016	0.00057621	0.013402
10000	5000	0.04573	0.00013672	0.0922143
10000	7500	0.04747	0.00012718	0.0921709



- The above graph supports more the previous talk about the time complexity for constant n entries and various Ks. Near to 500, the time complexity in the four cases becomes minimum and after that grows rabidly, specially for the worst case.

- This specific K represents the most suitable K for time complexity for merge/insertion sort, as it works like the balance point between both of them.

- At some point K, represented in 1000 for the above graph, the time grows rapidly and stabilizes after that.

Q2.1

d)

From B and C,

We can sum up that for a range of suitable small Ks, the time complexity is so small (from 0 to 1000 in the above graph). After that a specific point (near to 1000 in the previous one)—the threshold—the time grows rabidly.

Inside that range, there exists a minima k, at which the time complexity for all cases is the least, in the previous case :500—this minimum represents the best K .

However, if we graph the previous two graphs, we find that the suitable K changes according to the number of expected entries; but in both cases, the ratio K_{best} :n was the nearly the same. 50:1000 and 500:10000.

Q2.2

Problem 2.2 Recurrences

(a) $T(n) = 36T(n/6) + 2n$

using master method,

$\Theta(f(n)) = n^{\log_6 36} = n^2$, which implies

while $n^2 \geq 2n$ $\quad \lim_{n\to\infty} \frac{2n}{n^2} = 0$

then $T(n) \in \Theta(n^2)$ for $\varepsilon \geq 1$. $\quad T(n) \leq \Theta(n^{1+\varepsilon})$ $\square$

(b) $T(n) = 5T(n/3) + 17n^{1.2}$

using master method $\Theta(f(n)) = n^{\log_3 5} = n^{1.465}$

while $n^{1.465} \geq (17n^{1.2})$ $\forall n > 1$ $\quad \lim_{n\to\infty} \frac{17n^{1.2}}{n^{1.465}} = 0$

then $T(n) \in \Theta(n^{1.465})$ for $\varepsilon \geq 0.265$ $\quad T(n) \leq \Theta(n^{1.2+\varepsilon})$ $\square$

(c) $T(n) = 12T(n/2) + n^2 \lg n$

using master method $\Theta(f(n)) = n^{\log_2 12} = n^{3.584}$

while for $\forall n > 1$; $n^{3.584} > n^2 \log n$, $\lim_{n\to\infty} \frac{n^2 \log n}{n^{3.584}} = 0$

then $T(n) \in \Theta(n^{3.584})$ for $\varepsilon \geq 1$.

$T(n) \leq \Theta(n^{2+\varepsilon} \lg(n+3))$ $\square$

(d) $T(n) = 3T(n/5) + T(n/2) + 2^n$

2^n

$3(2^{n/5}) \qquad (2^{n/2}) \qquad 3(2^{n/5}) + 2^{n/2} \rightarrow 2^n(3(2^{-4n/5}) + 2^{-n/2})$

$9(2^{n/25}) \quad 3(2^{n/10}) \quad 3(2^{n/10}) \quad (2^{n/4}) \qquad 9(2^{n/25}) + 3(2^{n/10}) + 3(2^{n/10}) + 2^{n/4}$

$3(2^{n/5}) + 2^{n/2} + 9(2^{n/25}) + 3(2^{n/10}) + 3(2^{n/10}) + 2^{n/4} + \cdots$

$T(n) = 2^n\left(3(2^{-4n/5}) + 2^{-n/2} + 9(2^{-24n/25}) + 3(2^{-9n/10}) + 3(2^{-9n/10}) + 2^{-3n/4}\right) + \cdots$

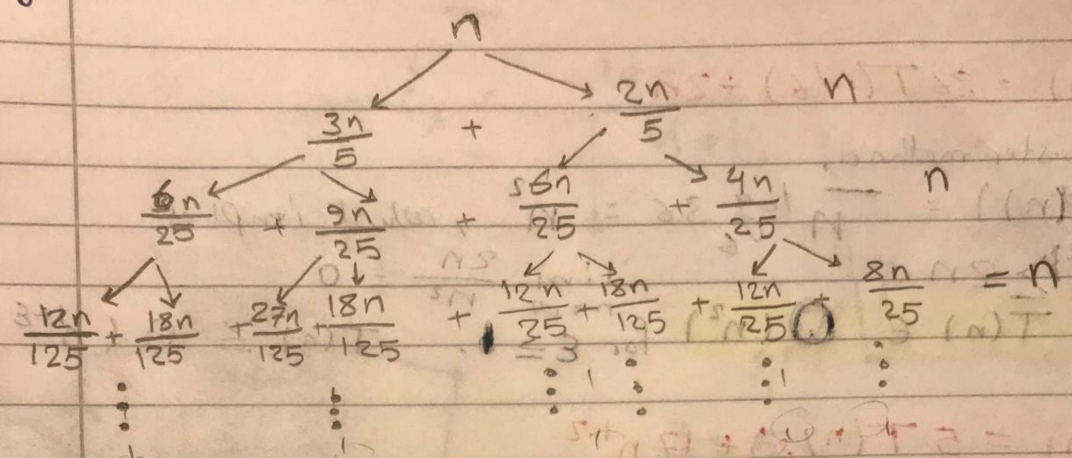
when $n \to \infty$ $T(n) = 2^n(3 + 1 + 9 + 3 + 3 + 1 + \cdots$

$2^n(4 + 16 + \cdots)$

So, $T(n) \in \Theta(2^n)$

(e) $T(n) = T(2n/5) + T(3n/5) + \Theta(n)$

using recurrence tree



$T(n) = n + n + n + \dots + n_{height}$

while height $= n \lg n$ and while $b = 2$

then so $T(n) \in \Theta(n \lg n)$