**Edwin Tomy George**

**ID: 80689715**

**Professor Ceberio**

**CS 2401 Elementary Data Structures**

**Friday, March 6th, 2020**

**Performance of Selection Sort**

*for (int i = 0; i < n - 1; ++i) {*

*int indexSmallest = i;*

*for (int j = i + 1; j < n; ++j) {*

*if (arr[j] < arr[indexSmallest]) {*

*indexSmallest = j;*

*}*

*}*

*int temp = arr[i];*

*arr[i] = arr[indexSmallest];*

*arr[indexSmallest] = temp;*

*}*

k1 is the constant time complexity before the nested for loop, k­­2 is the constant time complexity between the outer and inner loop, and k3 is the constant time complexity inside the inner loop. We have that:

T(K1) = 1

1. *int i = 0*

T(K2) = 7

1. *i < n – 1*
2. *i++*
3. *int indexSmallest = i*
4. *int j = i + 1*
5. *int temp = arr[i]*
6. *arr[i] = arr[indexSmallest]*
7. *arr[indexSmallest] = temp*

T(K3) best = 3

T(K3) worst = 4

T(K3) average = 3.5

1. *j < n*
2. *j++*
3. *if(arr[j] < arr[indexSmallest])*
   1. *worst case: if statement always true, T = 2*

*indexSmallest = j*

* 1. *best case: if statement always false, T = 1*
  2. *average case: averaging best and worst case, T = 1.5*

The total time complexity for the nested for loop, is k1 (outside the for loop), + (n – 1)k2 (for T(N) inside the outer for loop but outside the inner for loop) and for inside the inner loop:

So, for T(n) best, T(n) worst, T(n) average, we have

Then for all T(N), O(N) = n2.

**Performance of Insertion Sort**

*for (int i = 1; i < n; ++i) {*

*int j = i;*

*while (j > 0 && arr[j] < arr[j - 1]) {*

*int temp = arr[j];*

*arr[j] = arr[j - 1];*

*arr[j - 1] = temp;*

*--j*

*}*

*}*

k1 is the constant time complexity before the for loop, k­­2 is the constant time complexity between the for and while loop, and k3 is the constant time complexity inside the while loop. We have that:

T(K1) = 1

1. *int i = 1*

T(K2) = 3

1. *i < n*
2. *i++*
3. *int j = i*

T(K3) = 5

*1. while (j > 0 && arr[j] < arr[j - 1])*

1. *int temp = arr[i]*
2. *arr[i] = arr[indexSmallest]*
3. *arr[indexSmallest] = temp*
4. *j--*

The total time complexity for the nested for/while loop, is k1 (outside the for loop), (n – 1)k2 (for T(N) inside the outer for loop but outside the while loop). For the while loop, we have that j goes from i to 1, and i goes from 1 to n-1, so the total times the while loop passes it’s (n-1)n/2, multiplied by k3­, which is T(N) inside the while loop:

We have that for the best case, we never enter the while loop (arr[j] < arr[j - 1] is always false), the worst case is when we always enter, and the average case is between both.

So, for T(n) best, T(n) worst, T(n) average, we have

Then for average and worst-case T(N), O(N) = n2, and for best case T(N), O(N) = n2.

**Performance of Bubble Sort**

*for (int i = 0; i < n - 1; i++) {*

*for (int j = 0; j < n - i - 1; j++) {*

*if (arr[j] > arr[j+1]) {*

*int temp = arr[j]*

*arr[j] = arr[j + 1]*

*arr[j + 1] = temp*

*}*

*}*

*}*

k1 is the constant time complexity before the for loop, k­­2 is the constant time complexity between the for and while loop, and k3 is the constant time complexity inside the while loop. We have that:

T(K1) = 1

1. *int i = 0*

T(K2) = 3

1. *i < n – 1*
2. *i++*
3. *int j = 0*

T(K3)worst = 6

T(K3)best = 3

T(K3)average = 4.5

*1. j < n – i - 1*

1. *j++*
2. *if(arr[j] > arr[j + 1]*

*In the best-case scenario, the if statement is false, having only one line to follow. In the worst-case scenario, the if statement is true, having four lines to follow. The average is 2.5.*

* 1. *int temp = arr[j]*
  2. *arr[j] = arr[j + 1]*
  3. *arr[j + 1] = temp*

The total time complexity for the nested for loop, is k1 (outside the for loop), (n – 1)k2 (for T(N) inside the outer for loop but outside the inner loop). For the inner loop, we have that j goes from 0 to n – i – 1, and i goes from 0 to n – 1, so the total times the while loop passes it’s the sum from 0 to n – 1, multiplied by k3­, which is T(N) inside the while loop:

So, for T(n) best, T(n) worst, T(n) average, we have

Then for all T(N), O(N) = n2.

**Performance of Binary Search**

*int mid = 0;*

*int low = 0;*

*int high = n – 1;*

*while (high >= low) {*

*mid = (high + low) / 2;*

*if (arr[mid] < key) {*

*low = mid + 1;*

*}*

*else if (numbers[mid] > key) {*

*high = mid – 1;*

*}*

*else {*

*return mid;*

*}*

*}*

*return -1;*

k1 is the constant time complexity before the while loop, k­­2 is the constant time complexity inside the for loop. We have that:

T(K1) = 3

1. *int mid = 0*
2. *int low = 0*
3. *int high = n – 1*

T(K2) = 4

1. *high >= low*
2. *mid = (high + low) / 2*
3. *if else statement*
4. *low = mid + 1* or *high = mid – 1* or *return mid*

The total time complexity is k1 (outside the while loop), plus for inside the for loop, we have that that the difference between high and low, n, gets divided by halves constantly until reaching 0, which means finding how many times you can divide n by two, which is log2n. At the end for the worst case, you add one for return -1:

So, for T(n) best is when we find it at the first step inside the while statement, T(n) worst is when pass through everything, T(n) average is the average, so we have

Then for average and worst-case T(N), O(N) = n2, and for best case T(N), O(N) = logn.