**Time Complexity Analysis (29 April 2022)**

Time complexity of below code,

for(var i=0;i<n;i++)

    i\*=k

O(logkn)

int a = 0, i = N;

while (i > 0) {

    a += i;

    i /= 2;

}

O(log2 N) or O(log N)

int i, j, k = 0;

for (i = n / 2; i <= n; i++) {

    for (j = 2; j <= n; j = j \* 2) {

        k = k + n / 2;

    }

}

O(n log n)

---------------------------------------------------------

**Find the complexity of the recurrence:**

T(n) = { 2T(n-1) - 1, if n>0,

{ 1, otherwise

**Solution:**

Let us try solving this function with substitution.

T(n) = 2T(n-1) - 1

= 2(2T(n-2)-1)-1

= 22(T(n-2)) - 2 - 1

= 22(2T(n-3)-1) - 2 - 1

= 23T(n-3) - 22 - 21 - 20

.....

.....

= 2nT(n-n) - 2n-1 - 2n-2 - 2n-3

..... 22 - 21 - 20

= 2n - 2n-1 - 2n-2 - 2n-3

..... 22 - 21 - 20

= 2n - (2n-1)

[**Note:** 2n-1 + 2n-2 + ...... + 20 = 2n - 1]

T(n) = 1

Time Complexity is O(1). Note that while

the recurrence relation looks exponential

the solution to the recurrence relation

here gives a different result.

void function(int n)

{

    int i = 1, s =1;

    while (s <= n)

    {

        i++;

        s += i;

        printf("\*");

    }

}

O(sqrt(n))

Note: We can define the terms ‘s’ according to relation si = si-1 + i. The value of ‘i’ increases by one for each iteration. The value contained in ‘s’ at the ith iteration is the sum of the first ‘i’ positive integers. If k is total number of iterations taken by the program, then while loop terminates if: 1 + 2 + 3 ….+ k = [k(k+1)/2] > n So k = O(√n).  
Time Complexity of the above function O(√n).

Linear Search

------------------

Best Case T.C : O(1)

Average & Worst Case : O(n)

Merge Sort

-----------

Best & Worst Case Time Complexity : O(n log n)

Note: For Quick Sort (where pivot is chosen as a corner element), the worst occurs when the input array is already sorted, and the best occurs when the pivot elements always divide the array into two halves. For insertion sort, the worst case occurs when the array is reverse sorted, and the best case occurs when the array is sorted in the same order as output.

| **Algorithm** | **Time Complexity** | | |  |
| --- | --- | --- | --- | --- |
|  | **Best** | **Average** | **Worst** |  |
| [Selection Sort](http://geeksquiz.com/selection-sort/) | Ω(n^2) | θ(n^2) | O(n^2) |  |
| [Bubble Sort](http://geeksquiz.com/bubble-sort/) | Ω(n) | θ(n^2) | O(n^2) |  |
| [Insertion Sort](http://geeksquiz.com/insertion-sort/) | Ω(n) | θ(n^2) | O(n^2) |  |
| [Heap Sort](http://geeksquiz.com/heap-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) |  |
| [Quick Sort](http://geeksquiz.com/quick-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n^2) |  |
| [Merge Sort](http://geeksquiz.com/merge-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) |  |
| [Bucket Sort](https://www.geeksforgeeks.org/bucket-sort-2/) | Ω(n+k) | θ(n+k) | O(n^2) |  |
| [Radix Sort](https://www.geeksforgeeks.org/radix-sort/) | Ω(nk) | θ(nk) | O(nk) |  |
| [Count Sort](https://www.geeksforgeeks.org/counting-sort/) | Ω(n+k) | θ(n+k) | O(n+k) |  |