Bubble Sort:

At itiration 1:

Numer of comarisons = N-1

" " swap = N-1

At itiration 2:

" " " = N-2

" " " = N-2

⋮

At itiration N-1

" " " = 1

" " " = 1

calculate total:

$(N-1)+(N-2)+(N-3)+\cdots+2+1 = (N-1)*(N-1+1)/2$

N Natural Number formula

$= N*(N-1)/2 \Rightarrow T(n) = T(n)$* worst case comarissions and swaps

$= O(N^2)$ worst case

best case if we given sorted array it will be O(N) because we get N comarasions.

merge sort:

$T(N)$ = time taken to sort N elements

$M(N)$ = " " " merge N "

so its $\Rightarrow T(N) = 2 * T(N/2) + M(N) = 2 * T(N/2) + N$

$N/2$ element future divides so

$T(N) = 2 * [2T(N/4) + N/2) + N = 4 * T(N/4 + 2 * N \ldots$

$\ldots = 2^k * T(N/2^k) + k * N$

$N/2^k = 1$ because we divided until 1 $\Rightarrow k = \log_2 N$

$T(N) = N * T(1) + N * \log_2 N = N + N * \log_2 N$

so its $O(N \log N)$

Quick sort:

$T(N)$: time complex of k elements

$p(N)$: " " " finding position of pivot among k elements

Best case: $T(N) = 2 * T(N/2) + N$

$T(N) = 2 * (2 * T(N/4) + N/2) + N = 4T(N/4) + 2 * N$

$\Rightarrow T(N) = 2^k * T(N/2^k) + k * N \quad \Rightarrow 2^k = N \Rightarrow k = \log_2 N$

So $T(N) = N * T(1) + N * \log_2 N \Rightarrow O(N * \log N)$

Worst case:

$T(N) = T(N-1) + N = T(N-2) + (N-1) + N =$

$= T(N-2) + 2 * N = T(N-3) + 3 * N = \dots =$

$= T(N-k) + k * N - (k-1) - \dots - 2 = T(N-k) + k * N - k * (k-1)/2$

if $k = N \Rightarrow$

$\Rightarrow T(N) = T(0) + N * N - N(N-1)/2 = N^2 - N(N-1)/2$

$= N^2/2 + N/2 \Rightarrow O(N^2)$

(1) [redacted] space: complex

[redacted]

for each recursive call local variables we pushed onto the call stacks

The depth of callstacks is the number of recursive calls made

The space complex is O(n) due to maximum depth of call stack

(2) [redacted] space: complex

The depth of the call stack depends on the number of call recursive made

in this code the generate permutations the depth of call stack reaches the length input string factorial times if the string of length n the number of recursive calls and the depth of the call stack would be n!

The input space and space used for call stack determine the overall space complexity

The space complexity of generating permutions is O(n!) due to factorial growth in depth