DATE: __/__/____

Name: Danijal Saeed

Sap id: 53937

Section: BS Data Science

Course: Analysis of Algorithm

Assignment : 2

Problems

1) Linear Search

```
int linearSearch(int arr[], int n, int target){
  for(int i=0; i<n; i++){
   if(arr[i] == target){
     return i;
} }
    return -1; }
```

Explanation:

A single loop run from 0 to n-1.
Each iteration perform 1 compalation
Worst Case: All n elements checked
Best Case: fist at index 1

Calculation:

$T(n) = n$

$O(n)$

DATE: __/__/____

2) Binary Search

```
int binary search(int arr[], int left, int right, int
                                                target){
while(left <= right){
int mid = left + (right - left) / 2;
if(arr[mid] == target) return mid;
if(arr[mid] < target) left = mid + 1;
else right = mid - 1; }
return -1; }
```

Explanation:

Each iteration half the arr size
Max step = log n
Each step does constant comparison
Best case: found at first mid.

Calculation:

Interval after k step = $n/2^k$

$n/2^k \leq 1$

$T(n) = \lceil \log_2 n \rceil$

O(log n)

3) Bubble Sort:

```
void bubbleSort(int arr[], int n){
for(int i=0; i<n-1; i++){
for(int j=0; j<n-i-1; j++){
if(arr[j] > arr[j+1]{
std::swap(arr[j], arr[j+1]); }
} } }
```

Explanation:

Two nested loops

DATE: __/__/____

outer loop = 1+2+ ... +(n-1)

(n-1) n/2 = (n²-n)/2

T(n) = n²/2

O(n²)

4) **Selection Sort:**

```
void selectionSort(int arr[], int n){
 for(int i=0; i<n-1; i++){
 int minIndex = i;
 for(int j=1; j<n; j++){
 if(arr[j] < arr[minIndex]){
  minIndex = j; } }
 std::swap(arr[j], arr[minIndex]); }}
```

Explanation:

Outer loop runs n-1 times

Inner loop compare elements to find min.

Total comparisons ≈ (n-1)+(n-2)+... +1

Swap = n-1 (negligible)

Calculation:

Total comparison = (n-1)n/2 = (n²-n)/2

T(n) = n²/2

O(n²)

5) **Matrix Multiplication:**

```
void matrixMultiplication(int A[][N], int[][N],
                          int C[][N], int n){
for(int i=0; i<n; j++){
for(int j=0; j<n; j++){
  C[i][j] = 0;
```

DATE: __/__/____

```
for (int k = 0; k<n; k++){
C[i][j] += A[i][k] B[k][j]; }
} } }
```

Explanation:

Three nested loops so growth is cubic
each runs n time
work per step = 1 multiplication + 1 add

Calculation:

Iteration = $n \times n \times n = n^3$
Multiplication = n^3
Addition = $n^3 - n^2$
$T(n) = 2n^3$

$O(n^3)$

6) **Fibonacci (Recursive)**

```
int fibonacci(int n){
if(n<=1) return n;
return fibonacci(n-1) + fibonacci(n-2); }
```

Explanation:

each call generate 2 further call
Tree of recursive call
Height of tree = n
Growth is exponential

DATE : __/__/____

Sap id : 53937

Calculation:

Recurrence : $T(n) = T(n-1) + T(n-2) + 1$

Solution $\approx T(n) \approx \phi^n$, $\phi \approx 1.618$

upper bond : $O(2^n)$

$O(2^n)$

7) **Fibonacci (Iterative)**

```
int fibonacci (int n) {
if (n<=1) return n;
int a=0, b=1, c;
for(int i=2; i<n; i++) {
  c = a+b;
   a = b;
b = c; }
return b; }
```

Explanation:

Single loop from 2 to n

Each iteration does constant work

Linear growth with n

Same cost in best/worst cases.

Calculation :

Iteration = n-1

work per iteration = constant

$T(n) = n-1$

DATE: __/__/____

O(n)

1) Factorial (Recursive)

```
int factorial(int n){
if(n==0) return 1;
return n * factorial(n-1); }
```

Explanation:

Each call reduces n by 1
Total calls = n+1 (including base)
Each call does constant work
Linear growth

Calculation:

Recurrence: $T(n) = T(n-1) + 1$

$T(n) = n + 1$

O(n)

2) Nested Loops

```
void nestedLoops(int n){
for(int i=0; i<n; i++){
for(int j=0; j<n; j++){
std::cout<<i<<" "<<j<<std::endl;
}}}
```

DATE : __/__/____

Explanation:
Outer loop runs n times.
Inner loop also runs n times
Total iterations = n x n
Growth is quadratic

Calculation:
Iteration = $n \times n = n^2$
Each iteration = constant printing
$T(n) = n^2$

$O(n^2)$

10) **Recursive Power Function**

```
int power(int x, int n) {
if(n==0) return 1;
return x * power(x, n-1); }
```

Explanation:
Each call decreases n by 1
Total calls = n+1 (including base)
Each call does constant multiplication
linear growth

Calculation:
Recurrence. $T(n) = T(n-1) + 1$
$T(n) = n+1$

$O(n)$