Algorithm Lab 4

Supta Philip Philip

October 17, 2022

1 Pre-Requisite in C++

1. Function
2. Pointer
3. Array / Dynamic Array
4. Dynamic Array - 2D/ Matrix
5. Random number
6. File Operation
7. Standard Template Library(STL)

2 Recursive Function

1. Fibonacci
2. Factorial
3. Euclid's algorithm for GCD

Algorithm 1 Fibonacci

```
1: procedure FIBONACCI(n)
2:     if n ≤ 1 then
3:         return n
4:     else
5:         return FIBONACCI(n − 1)+FIBONACCI(n − 2)
6:     end if
7: end procedure
```

Algorithm 2 Factorial

```
1: procedure FACTORIAL(n)
2:     if n ≤ 0 then
3:         return 1
4:     else
5:         return n*FACTORIAL(n − 1)
6:     end if
7: end procedure
```

Algorithm 3 Euclid's algorithm[Recursive]

```
1: procedure GCD(n, m)
2:     if n = m then
3:         return m
4:     else if n ≥ m then
5:         return GCD(n − m, m)
6:     else
7:         return GCD(n, m − n)
8:     end if
9: end procedure
```

Algorithm 4 Euclid's algorithm for GCD[Iterative]

```
1: procedure GCD(a, b)                          ▷ The g.c.d. of a and b
2:     r ← a mod b
3:     while r ≠ 0 do                           ▷ We have the answer if r is 0
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     end while
8:     return b                                 ▷ The gcd is b
9: end procedure
```

3 Iterative Algorithm

1. Linear Search
2. Bubble sort
3. Selection sort
4. Insertion Sort

Algorithm 5 Linear Search

```
1: procedure LINEAR(A, n, item)
2:     for i ← 0, n − 1 do
3:         if A[i] == item then
4:             return i
5:         end if
6:     end for
7:     return −1
8: end procedure
```

Algorithm 6 Bubble Sort

```
1: procedure BUBBLESORT(A, n)
2:     for k ← 0, n − 1 do
3:         for i ← 0, n − 1 do
4:             if A[i] > A[i + 1] then
5:                 swap(A[i], A[i + 1]
6:             end if
7:         end for
8:     end for
9: end procedure
```

Algorithm 7 Selection Sort

```
1: procedure SelectionSort(A, n)
2:     for i ← 0, n − 1 do
3:         iMin ← i
4:         for j ← i + 1, n − 1 do
5:             if A[j] < A[iMin] then
6:                 iMin = j
7:             end if
8:             swap(A[iMin], A[i])
9:         end for
10:    end for
11: end procedure
```

Algorithm 8 Insertion Sort

```
1: procedure InsertionSort(A, n)
2:     for j ← 1, n − 1 do
3:         value ← A[j]
4:         i ← j − 1
5:         while i > 0&A[i] > value do
6:             swap(A[i], A[i + 1])
7:             i ← i − 1
8:         end while
9:         A[i + 1] = value
10:    end for
11: end procedure
```

4 Divide and Conquer Algorithm

Algorithm 9 Binary Search Recursive algorithm

```
1: procedure BinarySearch(A, low, high, x)
2:     if low > high then
3:         return −1
4:     end if
5:     mid = (low + high)/2
6:     if x == A[mid] then
7:         return mid
8:     else if x < A[mid] then
9:         return BinarySearch(A, low, mid − 1, x)
10:    else
11:        return BinarySearch(A, mid + 1, high, x)
12:    end if
13: end procedure
```

Algorithm 10 Binary Search Iterative algorithm

```
1: procedure BinarySearch(A, n, x)
2:     low = 0, high = n − 1
3:     while low < high do
4:         mid = (low + high)/2
5:         if x == A[mid] then
6:             return mid
7:         end if
8:         if x < A[mid] then
9:             high = mid − 1
10:        else
11:            low = mid + 1
12:        end if
13:    end while
14:    return −1
15: end procedure
```

Algorithm 11 Merge Two Arrays

```
1: procedure MERGE(A, B, n, m)
2:     i ← 0, j ← 0, k ← 0
3:     while i ≤ n − 1&j ≤ m − 1 do
4:         if A[i] > B[j] then
5:             C[k + +] ← B[j + +]
6:         else
7:             C[k + +] ← A[i + +]
8:         end if
9:     end while
10:    while i <= n − 1 do
11:        C[k + +] ← A[i + +]
12:    end while
13:    while j <= m − 1 do
14:        C[k + +] ← B[j + +]
15:    end while
16: end procedure
```

Algorithm 12 Merge

```
1: procedure MERGE(A, left, mid, right)
2:     n1 = mid − left + 1
3:     n2 = right − mid
       L[1....n1]andR[1....n2]
4:     for i ← 0, n1 − 1 do
5:         L[i] ← A[left + i]
6:     end for
7:     for j ← 0, n2 − 1 do
8:         R[j] ← A[mid + 1 + j]
9:     end for
10:    i ← 0, j ← 0, k ← left
11:    while i ≤ n1 − 1&j ≤ n2 − 1 do
12:        if L[i] < R[j] then
13:            A[k + +] ← L[i + +]
14:        else
15:            A[k + +] ← R[i + +]
16:        end if
17:    end while
18:    while i <= n1 − 1 do
19:        A[k + +] ← L[i + +]
20:    end while
21:    while j <= n2 − 1 do
22:        A[k + +] ← R[j + +]
23:    end while
24: end procedure
```

Algorithm 13 Merge Sort

```
1: procedure MergeSort(A, left, right)
2:     if left < right then
3:         mid = (left + right)/2
4:         MergeSort(A, left, mid)
5:         MergeSort(A, mid + 1, right)
6:         Merge(A, left, mid, right)
7:     end if
8: end procedure
```

Algorithm 14 partition

```
1: procedure partition(A, start, end)
2:     pivot = A[end]
3:     pIndex = start
4:     for i ← start, end − 1 do
5:         if A[i] < pivot then
6:             swap(A[i], A[pIndex])
7:             pIndex + +
8:         end if
9:     end for
10:    swap(A[pIndex], A[end])
11:    return pIndex
12: end procedure
```

Algorithm 15 Quick Sort

```
1: procedure QuickSort(A, start, end)
2:     if start >= end then
3:         pIndex =partition(A, start, end)
4:         QuickSort(A, start, pIndex − 1)
5:         QuickSort(A, pIndex + 1, end)
6:     end if
7: end procedure
```