# Class 2 Homework

## Chapter 3.1 Page 102, Questions 4,8,11

### Question 4

1. **Design a brute-force algorithm for computing the value of a polynomial**

at a given point and determine its worst-case efficiency class.

A brute force algorithm could look something like this with an algorithm of O(n^2)

// Brute-force method for polynomial evaluation

public static double evaluateBruteForce(double[] coefficients, double x0) {

double result = 0;

int n = coefficients.length - 1; // Degree of the polynomial

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

result += coefficients[i] \* Math.pow(x0, n - i); // a\_i \* x0^(n-i)

}

return result;

}

public static void main(String[] args) {

double[] coefficients = {1, -3, 2}; // Polynomial: p(x) = x^2 - 3x + 2

double x0 = 2.0; // Point at which to evaluate the polynomial

double result = evaluateBruteForce(coefficients, x0);

System.out.println("Polynomial evaluated at x = " + x0 + ": " + result);

}

}

The coefficients {1, -3, 2} represent the polynomial p(x)=x2−3x+2p(x) = x^2 - 3x + 2p(x)=x2−3x+2, where:

* a2=1
* a1=−3
* a0=2

The algorithm computes p(x0) by directly evaluating each term in the polynomial.

1. **If the algorithm you designed is in (n^2), design a linear algorithm for this problem.**

Horner's Method for p(x)=x^2−3x+2

public class Polynomial {

// Horner's method for polynomial evaluation

public static double evaluateHorner(double[] coefficients, double x0) {

double result = coefficients[0];

for (int i = 1; i < coefficients.length; i++) {

result = result \* x0 + coefficients[i]; // Horner's method

}

return result;

}

public static void main(String[] args) {

double[] coefficients = {1, -3, 2}; // Polynomial: p(x) = x^2 - 3x + 2

double x0 = 2.0; // Point at which to evaluate the polynomial

double result = evaluateHorner(coefficients, x0);

System.out.println("Polynomial evaluated at x = " + x0 + ": " + result);

}

}

This method uses Horner’s scheme to evaluate p(x0) with fewer operations, as shown in the formula:

p(x)=(x0⋅(x0 −3))+2

Horner's method will compute the result by iteratively updating the result.

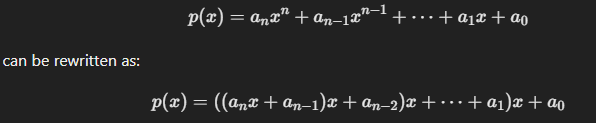
Polynomial evaluated at x = 2.0: 0.0

p(2)=(2^2)−3(2)+2=4−6+2=0

1. **Is it possible to design an algorithm with a better-than-linear efficiency for this problem?**

A more efficient way to evaluate the polynomial is to use **Horner’s method**. This method rewrites the polynomial to reduce the number of operations involved in computing powers of x0​.

Using Horner's method, the polynomial:



This allows us to compute the polynomial in a more efficient manner, avoiding the need to compute powers of x0​ directly.

Here is the implementation of Horner’s method:

def polynomial\_horner(coefficients, x\_0):

result = coefficients[0]

for i in range(1, len(coefficients)):

result = result \* x\_0 + coefficients[i]

return result

**Efficiency of Horner’s Method**

In this algorithm, for each coefficient ai, we perform one multiplication and one addition. Since there are n coefficients (for a polynomial of degree n−1), the total number of operations is O(n).

Thus, the time complexity of this linear algorithm is O(n).

**Possibility of a Better-Than-Linear Algorithm**

Since polynomial evaluation is a problem where the number of operations is inherently tied to the degree of the polynomial (i.e., n terms to sum), it is unlikely that there is an algorithm that evaluates the polynomial in less than O(n) time. Each term of the polynomial needs to contribute to the final result, and there is no shortcut that eliminates the need for considering all the terms. Thus, a **linear-time** algorithm is optimal for this problem, and **no algorithm with better-than-linear efficiency** (i.e., less than O(n)) is possible.

Overall

* + **Brute-force algorithm** has a time complexity of O(n2)
  + **Horner's method** is a linear-time algorithm with O(n) complexity, which is the best we can achieve.
  + A better-than-linear time complexity algorithm is not possible for this problem.

### Question 8

Sort the list E, X, A, M, P , L, E in alphabetical order by selection sort

public class SelectionSort {

public static void main(String[] args) {

// Initialize the array

char[] arr = {'E', 'X', 'A', 'M', 'P', 'L', 'E'};

// Perform selection sort

selectionSort(arr);

// Print the sorted array

System.out.print("Sorted array: ");

for (char c : arr) {

System.out.print(c + " ");

}

}

// Selection Sort method

public static void selectionSort(char[] arr) {

int n = arr.length;

// Move through the entire array

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

// Find the minimum element in the unsorted part of the array

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j; // Update minIndex if a smaller element is found

}

}

// Swap the found minimum element with the first element of the unsorted part

char temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

}

The array will be sorted in alphabetical order as A, E, E, L, M, P, X.

### Question 11

Sort the list E, X, A, M, P , L, E in alphabetical order by bubble sort.

public class BubbleSort {

public static void main(String[] args) {

// Initialize the array

char[] arr = {'E', 'X', 'A', 'M', 'P', 'L', 'E'};

// Perform bubble sort

bubbleSort(arr);

// Print the sorted array

System.out.print("Sorted array: ");

for (char c : arr) {

System.out.print(c + " ");

}

}

// Bubble Sort method

public static void bubbleSort(char[] arr) {

int n = arr.length;

// Bubble sort algorithm

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

// Last i elements are already sorted, so no need to check them

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

// If the current element is greater than the next, swap them

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

// Swap the elements

char temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

}

## Chapter 3.1 page 107 Question 5,8

## Question 5

How many comparisons (both successful and unsuccessful) will be made by the brute-force algorithm in searching for each of the following patterns in the binary text of one thousand zeros?

1. 00001

Total number of positions to check: 1000−5+1=996

All comparisons that will be unsuccessful 996×5=4980

1. 10000

Total number of positions to check: 1000−5+1=996

All comparisons that will be unsuccessful 996×5=4980

1. 01010

Total number of positions to check: 1000−5+1=996

All comparisons that will be unsuccessful 996×5=4980

## Question 8

Consider the problem of counting, in a given text, the number of substrings that start with an A and end with a B. For example, there are four such substrings in CABAAXBYA

1. Design a brute-force algorithm for this problem and determine its efficiency class.

public class SubstringCounter {

// Brute-force approach to count substrings starting with 'A' and ending with 'B'

public static int countAtoBSubstrings(String text) {

int count = 0;

int n = text.length();

// Loop through all possible starting positions (i)

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

if (text.charAt(i) == 'A') { // Find 'A' at position i

// Loop through all possible ending positions (j)

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

if (text.charAt(j) == 'B') { // Find 'B' at position j

count++; // Increment count when a valid substring is found

}

}

}

}

return count; // Return the total count of valid substrings

}

public static void main(String[] args) {

String text = "CABAAXBYA";

int result = countAtoBSubstrings(text);

System.out.println("Number of substrings starting with 'A' and ending with 'B': " + result);

}

}

Explanation of Brute-Force Approach:

* The outer loop goes through all positions in the string. For each position iii where we find 'A', the inner loop checks every subsequent position jjj for 'B'
* Each valid combination of iii and jjj where iii points to an 'A' and jjj points to a 'B' is counted as a valid substring.

Time Complexity of Brute-Force: The time complexity is O(n2)O(n^2)O(n2), where nnn is the length of the string because for each 'A', we check all the positions after it for 'B'.

1. Design a more efficient algorithm for this problem. [Gin04]

public class SubstringCounter {

// Optimized approach to count substrings starting with 'A' and ending with 'B'

public static int countAtoBSubstringsOptimized(String text) {

int countA = 0; // To keep track of the number of 'A's encountered

int totalCount = 0; // To count the number of valid substrings

// Loop through the string once

for (int i = 0; i < text.length(); i++) {

char currentChar = text.charAt(i);

if (currentChar == 'A') {

countA++; // Increment count when 'A' is encountered 1,2,3, 5

} else if (currentChar == 'B') {

totalCount += countA; // Add countA to totalCount when 'B' is encountered 4

}

}

return totalCount; // Return the total count of valid substrings

}

public static void main(String[] args) {

String text = "CABAAXBYA";

int result = countAtoBSubstringsOptimized(text);

System.out.println("Number of substrings starting with 'A' and ending with 'B': " + result);

}

}