Design Document Assignment 2

Abhinav Prasanna

October 15, 2021

Introduction

The core concept of this assignment is to understand and create sorting algorithms with the provided algorithms in our PDF. We must code the following sorting algorithms: Insertion Sort, Heap Sort, Quick Sort, and Shell Sort. We must use the provided declaration for these methods and implement it in C. We must also create a test class that supports flags for different sorting methods. We must also create a set.h file to track which command-line options are specified when the program is to run.

insert.c

insert.c is a program responsible for completing insertion sort. Insertion Sort considers the elements one at a time and sorts them in order. Insertion Sort compares the k-th element with each preceeding element in descending order till the position is found.

We sort the array A in the right in increasing order. We start to check for A[k] and check with A[k-1] and see if it is in the same order.

If A[k] is in the right place, this means if A[k] is greater than A[k-1] then we can move on to sort the next element.

If A[k] is in the wrong place, we must sort A[k] into A[k-1] meaning A[k] is less than A[k-1] and we shall flip the elements in order. Then we move onto the next element.

Pseudocode

```
void insertionsort(int* A){
int j,temp;
   for I in A.length{
     j=I
     temp=A[I]
   }
   while j>0 and temp<A[j-1]{
   A[j]=A[j-1]
   j-=1
   }
   A[j]=temp
}
```

shell.c

Shell.c is a program responsible for running shell sort which is variation of insertion sort. Shell Sort will sort the first pair of elements which are far apart each other. The distance of elements is called gap. Each iteration of shell sort decreases the gap by 1. Shell Sort has a complexity time of $O(n^{\frac{5}{3}}$

Pseudocode

```
void gaps(int n){
  for int i in range of log(3+2*n) / log(3),0,1 {
  yield( 3 * *i -1)//2
  }
}
void shellsort(
```

Euler.c

Euler.c is a program responsible for approximating π using the Euler Series. We also need a function to track the number of computed terms with a static variable. $\sum_{k=1}^{\infty} \frac{1}{k!}$
We need the functions pieuler() and pieulerterms() where pieuler approximates

the value of π using Euler series and pieulerterms tracks the number of computed terms with a static variable.. We can use a for loop to take a summation but we will need a finite term to end. This is where we need to check if the last term converges onto ϵ.

Pseudocode

```
for (int k; lastterm > epsilon ; k++){
     lastterm = 1/k * k
     sum += lastterm
     terms++
}
return sum and terms
```

bbp.c

BBP.c is a program responsible for approximating π using the Bailey-Borwein-Plouffe Formula. We also need a function to track the number of computed terms with a static variable. $\sum_{k=0}^{\infty} 16^{-k} \frac{k(120k+151)+47}{(k(k(k(512k+1024)+712)+194)+15)}$
We need the functions pibbp() and pibbpterms() where pibbp approximates the value of π using BBP series and pibbpterms tracks the number of computed terms with a static variable. We can use a for loop to take a summation but we will need a finite term to end. This is where we need to check if the last term converges onto ϵ. We need to use a for loop for power function.

Pseudocode

```
for (int k; lastterm > epsilon ; k++){
    for(int j;j<k;j++){
         power*=16;
    }
    coefficient = 1/power
    numerator = k * ( 120 * k + 151) + 47
    denomonator = k * ( k * ( k * ( 512 * k + 1024) + 712 ) + 194 ) + 15
    lastterm = coefficient * ( numerator / denomonator)
    sum+= lasterm
    terms++
}
return sum and terms
```

viete.c

Viete.c is a program responsible for approximating π using the Viete Formula. We also need a function to track the number of computed terms with a static variable. $\prod_{i=1}^{\infty} \frac{ai}{2}$ We need the functions piviete() and pivieteterns() where piviete approximates the value of π using Viete series and pivieteterms tracks the number of computed terms with a static variable.. We can use a for loop to take a summation but we will need a finite term to end. This is where we need to check if the last term converges onto ϵ.

Pseudocode

```
for (int k; lastterm > epsilon ; k++){
    lasterm = k/2
    sum *= lasterm
    terms++
}
return sum and terms
```

newton.c

Viete.c is a program responsible for approximating the Newton-Raphson method and track the number of iterations taken. We also need a function to track

the number of computed terms with a static variable.. $x_{k+1} = x_{k+1} - \frac{f(x_k)}{f'(x_k)}$
We need the functions sqrtnewton() and sqrtnewtonitters() where sqrtnewton approximates the value of π using Newton-Raphson method and sqrtnewtoniters tracks the number of computed terms with a static variable. We can use the sqrt function below to emulate Newton- Raphson Method.

PseudoCode

```
sqrt(x)
x=0
y=1
 while(abs(y-z) > EPSILON
   z=y
   y = 0.5 * (z+x/z)
 return y
```

mathlib-test.c

MathLib-test.c contains different test cases for our c files. The a flag should run all tests. The e flag should run approximation tests. The b flag should run Bailey Borwein Plouffe π approximation test. The m flag runs Madhava π approximation test. The r flag runs Euler sequence π approximation test. The v flag runs Viete sequence π approximation test. The n flag runs Newton-Raphson Square Root approximation tests. The s flag enable printing of statistics to see computed terms and factors for each tested function. The h flag displays a help message detailing program usage.

PseudoCode

```
switch
case(a)
run tests
case(e)
run approximation tests
case(b)
run bbp.c
case(m)
run madhava.c
case(r)
run euler.c
case(v)
run viete.c
case(n)
run newton.c
case(s)
print all c files
case(h)
print help message
```