Adhesh Reghu Kumar
COE18B001

Operating Systems Lab

Lab Assignment - 3

2.(a) Odd and Even series generation for n terms using Parent Child relationship (say odd is the duty of the parent and even series as that of child)

Logic Used

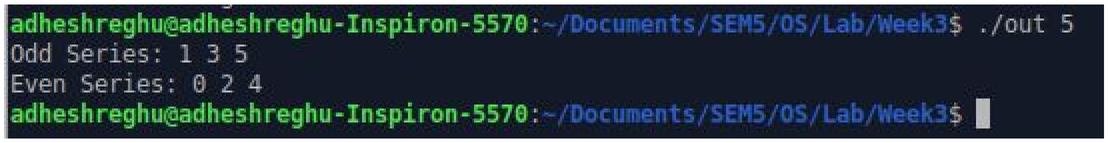
The program requires the concurrent generation of odd and even series less than input *n* in two processes - odd series in parent and even series in child. Since there is no need for data sharing, the processes can be independently executed in two concurrent processes generated using *fork()* .
First we accept the input ***n*** which will act as the upper limit for both the series through the command line. In the parent process (pid > 0), we generate the odd series less than (or equal to) n using a for loop and mod operator, similarly we generate the even series less than (or equal to) n in the child process (pid = 0).

Code

```c
int main(int argc, char *argv[])
{
    if(argc < 2)
       printf("Insufficient arguments\n");
    else if(argc == 2)
    {
       int n = atoi(argv[1]);
       pid_t pid = fork();                // Create child process
       if(pid == 0)
       {
             int i=0;
             printf("Even Series: ");
             for(;i<=n;++i)
                   if(i%2 == 0)
                         printf("%d ",i);
             puts("");
             exit(0);
       }
```

```
        else
        {
                int i=0;
                printf("Odd Series: ");
                for(;i<=n;++i)
                        if(i%2 == 1)
                                printf("%d ",i);
                puts("");
        }
    }
    else
        printf("Too many arguments.\n");
    return 0;
}
```

Output:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 5
Odd Series: 1 3 5
Even Series: 0 2 4
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$
```

2.(b) given a series of n numbers (u can assume natural numbers till n) generate the sum of odd terms in the parent and the sum of even terms in the child process.

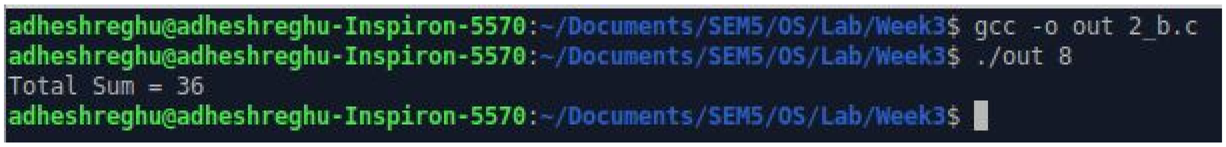
Logic Used

The program requires the calculation of the sum of first n natural numbers by concurrently computing the sum of odd series in the parent and sum of even series in the child process.
In this program, there is sharing of data, hence we use *vfork()*. We share the *sum* variable which gets updated with even series sum in the child process and odd series sum in the parent. We terminate the child process after computing the even series sum using *exit(0)*. At the end, the parent prints the final sum.
The input n is passed as a command line argument.

Code

```c
int main(int argc, char *argv[])
{
    if(argc < 2)
       printf("Insufficient arguments\n");
    else if(argc == 2)
    {
       int sum  = 0;
       int n = atoi(argv[1]);
       pid_t pid = vfork();                  // create child
       if(pid == 0)
       {
             int i=0;
             for(;i<=n;++i)
                   if(i%2 == 0)
                         sum = sum + i;
             exit(0);
       }
       else                   // compute odd series sum in parent.
       {
             int i=0;
             for(;i<=n;++i)
                   if(i%2 == 1)
                         sum = sum + i;
       }
       printf("Total Sum = %d\n",sum);
    }
    else
       printf("Too many arguments.\n");
    return 0;
}
```

Output:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ gcc -o out 2_b.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 8
Total Sum = 36
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ 
```

3. Armstrong number generation within a range. The digit extraction, cubing can be the responsibility of the child while the checking for sum == no can happen in the child and the output list in the child.

Logic Used

In this program we have to generate Armstrong numbers less than (or equal to) ***n***. We loop until *n* and perform these two operations

a. Extract digits and cube the digits and sum them up
b. Compare if the sum = number

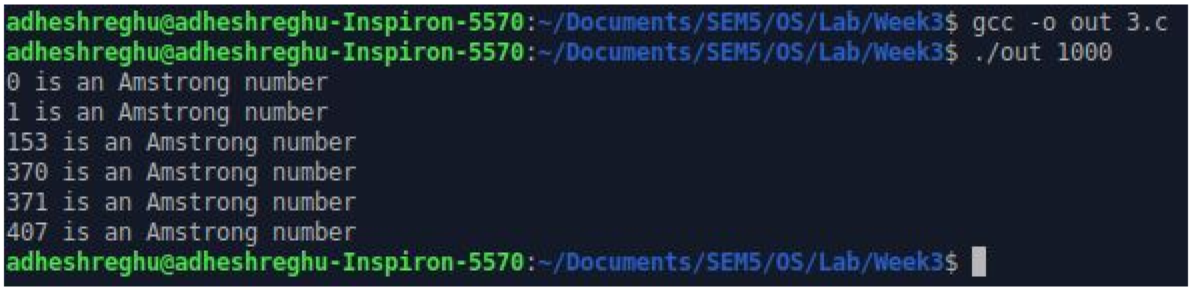
If the sum is the same as the number, the given number is an Armstrong number. The program requires us to execute (a) in child and (b) in parent process.
Since there is a need to share data (sum of cubed digits) between the parent and child process, we use *vfork().* We also have to wait for the child process to complete before executing the parent process, hence we use *wait(NULL)* immediately after forking to execute the child process first. We terminate the child process using *exit(0)*. In the parent process we use the *sum* that was computed in the child process to verify if the given number is an Armstrong number.
The input n is passed as a command line argument.

Code

```
int main(int argc, char *argv[])
{
    if(argc < 2)
       printf("Insufficient arguments\n");
    else if(argc == 2)
    {
       int n = atoi(argv[1]);
       int i =0;
       for(;i<=n;++i)
       {
             int cube_i =0;
             pid_t pid = vfork();   // fork a child and wait for it to complete
             wait(NULL);
```

```
            // child process - do digit extraction and cubing
            if(pid == 0)
             {
                   int num = i,digit=0;
                   while(num!=0)
                   {
                         digit = num%10;
                         cube_i = cube_i + (digit * digit * digit);
                         num = num/10;
                   }
                   exit(0);
             }
             else       // parent process - do check
             {
                   if(cube_i == i)
                         printf("%d is an Amstrong number\n",i);
             }
       }
    }
    else
       printf("Too many arguments.\n");
    return 0;
}
```

Output:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ gcc -o out 3.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 1000
0 is an Amstrong number
1 is an Amstrong number
153 is an Amstrong number
370 is an Amstrong number
371 is an Amstrong number
407 is an Amstrong number
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$
```

4. Fibonacci Series AND Prime parent child relationship (say parent does fib Number generation using series and child does prime series).

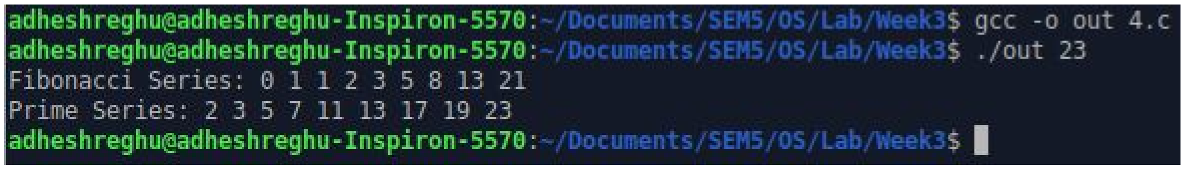
Logic Used

In this program we have to generate Fibonacci series and Prime series less than (or equal to) ***n***. We are required to generate the Fibonacci series in the parent process (pid>0) and the Prime series in the child process (pid = 0).
Since there is no need to share data, we use simple *fork().*
Inside the child block (pid=0), we compute fibonacci series in a bottom up fashion until *fib[i] <= n* . We then print the series.
In the parent block (pid > 0), we print all prime numbers less than (or equal to) n. Here we loop from 1 to n and check if the number is prime or not. If it is a prime, we print it else we continue our loop. The prime check is done by *isPrime()* function.

Code

```
#define MAX 1000
int isPrime(long int n);

int main(int argc, char *argv[])
{
    if(argc < 2)
       printf("Insufficient arguments\n");
    else if(argc == 2)
    {
       // have to validate the string
       int sum = 0;
       int n = atoi(argv[1]);
       pid_t pid = fork();
       if(pid == 0)
       {
             // Do the Prime series generation
             int i=1;
             printf("Prime Series: ");
             for(;i<=n;++i)
                   if(isPrime(i))
                         printf("%d ",i);
             puts("");
             exit(0);
       }
       else
       {
             // Do the fibonacci Series generation upto nth term
             int fib[MAX] = {0};
             fib[0] = 0;
             fib[1] = 1;
             int i=1;
             while(fib[i] <= n)
             {
                   fib[i+1] = fib[i] + fib[i-1];
                   i = i+1;
             }
             int j=0;
             printf("Fibonacci Series: ");
             for(;j<i;++j)
                   printf("%d ",fib[j]);
             puts("");
       }
    }
    else
       printf("Too many arguments.\n");
    return 0;
}

int isPrime(long int n)
{
      // Corner cases
      if (n <= 1)  return 0;
      if (n <= 3)  return 1;

      // This is checked so that we can skip
      // middle five numbers in below loop
      if (n%2 == 0 || n%3 == 0) return 0;

      int i = 5;
      for (; i*i<=n; i=i+6)
      if (n%i == 0 || n%(i+2) == 0)
            return 0;

      return 1;
}
```

Output:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ gcc -o out 4.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 23
Fibonacci Series: 0 1 1 2 3 5 8 13 21
Prime Series: 2 3 5 7 11 13 17 19 23
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$
```

5. Ascending Order sort within Parent and Descending order sort (or vice versa) within the child process of an input array. (u can view as two different outputs –first entire array is asc order sorted in op and then the second part desc order output)

Logic Used

In this program we have to sort an array in descending order in the parent process and ascending order in the child process. We accept the integer array through the command line. Since there is once again no need for data sharing, we use simple *fork().*
After integer validation, we begin our concurrent process execution. A standard sort function has been defined to sort an input array in any given order.
In the child process (pid = 0), we call the sort function and pass the input array and order = 1 (ascending). The child process is terminated with an *exit(0).*
In the parent process (pid > 0), we call the sort function and pass the input array and order = 0 (descending).
The sorted array gets printed within the *sortArray()* function.

Code:

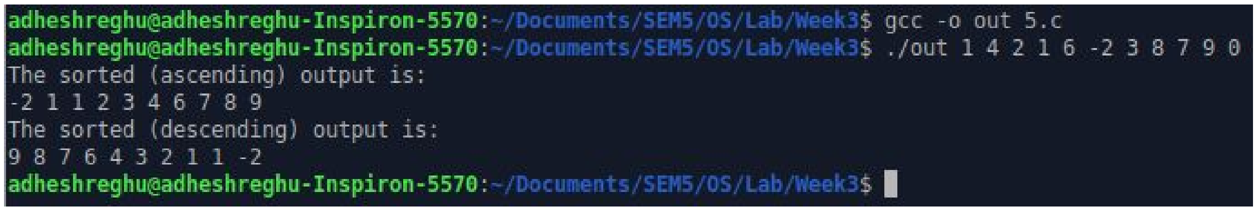
```
#define MAX 1000

void sortArray(int arr[],int n,int order);
int isInteger(char* string);

int main(int argc, char *argv[])
```

```
{

    if(argc < 2)
       printf("Insufficient arguments.\n");
    else
    {
       int len = argc - 1;
       int arr[MAX],n=0,order=1;
       int i=1;
       for(;i<len;++i)
       {
       // Validation function. Warning if not integer.
       if(isInteger(argv[i]))
             arr[n++] = atoi(argv[i]);
       else
             printf("warning: '%s' is not an integer. will be
ignored.\n",argv[i+1]);
       }

       // Warning if entire argument list is invalid.s
       if(n == 0 )
             printf("The input list is invalid. Please enter an integer array.\n");
       else
       {
             // Sort two ways - Desc in child and ascending in parent
             pid_t pid = fork();
             if(pid == 0)
             {
                   // Within child - sort in descending order. order = 0;
                   sortArray(arr,n,0);
                   exit(0);
             }


             else
             {
                   // sort descending within the parent process.
                   sortArray(arr,n,order = 1);
             }
       }
    }
    return 0;
```

```
}

void sortArray(int arr[],int n,int order)
{
    // For swap purposes
    int temp;

    // Initiate Loops for simple Bubble Sort.
    int i=0;
    for(;i<n;++i)
    {
       int j=0;
       for(;j<n-i-1;++j)
       {
             if(((order == 1) && (arr[j]>arr[j+1])) || ((order == 0) &&
(arr[j]<arr[j+1])))
             {
                   // Swap arr[j] and arr[j+1]
                   temp = arr[j];
                   arr[j] = arr[j+1];
                   arr[j+1] = temp;
             }
       }
    }
    // Print the sorted list.
    if(order == 1)
       printf("The sorted (ascending) output is:\n");
    else
       printf("The sorted (descending) output is:\n");
    i=0;
    for(;i<n;++i)
       printf("%d ",arr[i]);
    puts("");
}

// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
       character = str[i];
```

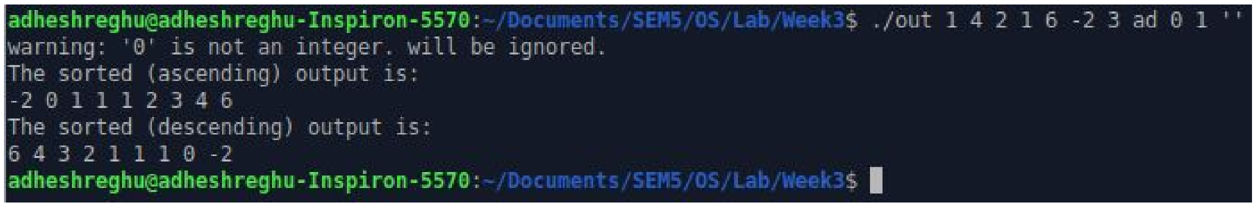
```
        // In case of '-'' symbol, continue if not the last character in the string.
        if(character == 45 && i<strlen(str)-1)
              continue;
        // Else get the decimal value (-48) and check condition.
        character -= 48;
        if(character < 0 || character > 9)
              return 0;
    }
    return 1;
}
```

Output:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ gcc -o out 5.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 1 4 2 1 6 -2 3 8 7 9 0
The sorted (ascending) output is:
-2 1 1 2 3 4 6 7 8 9
The sorted (descending) output is:
9 8 7 6 4 3 2 1 1 -2
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ 
```

With invalid entries mixed:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 1 4 2 1 6 -2 3 ad 0 1 ''
warning: '0' is not an integer. will be ignored.
The sorted (ascending) output is:
-2 0 1 1 1 2 3 4 6
The sorted (descending) output is:
6 4 3 2 1 1 1 0 -2
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ 
```

6. Given an input array use parent child relationship to sort the first half of array in ascending order and the trailing half in descending order (parent / child is ur choice)

Logic Used

In this program we have to sort the first half of an input array in ascending order and the remaining half in descending order.
In this program we will print the resultant array at the end after both the sorts and hence the child process will have to share its output with the parent. We can concurrently execute both the sorts with data sharing using v*fork()*.

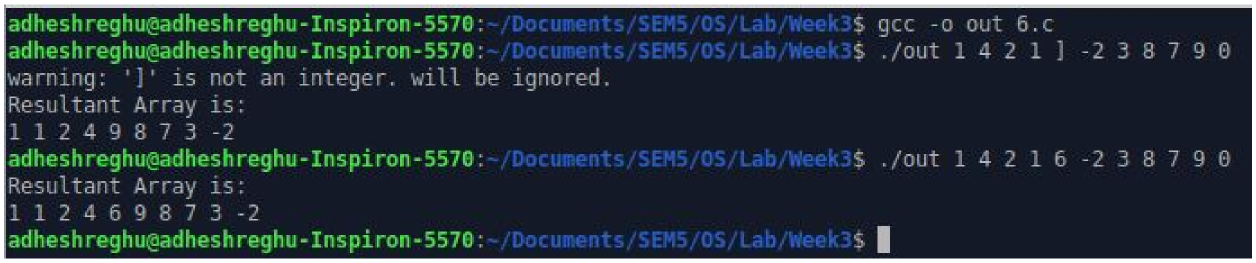
We accept the integer array through the command line. After integer validation, we begin our concurrent execution of the child and parent processes.
In the child process (pid = 0), we call the *sortArray()* to sort the input array indexed from *0* to *n/2* in ascending order (order = 1).
In the parent process (pid > 0), we call the *sortArray()* to sort the input array indexed from *n/2* to *n* in descending order (order = 0).
At the end of both processes, we print the resultant array.

Code:

```
#define MAX 1000

void sortArray(int arr[],int start,int end,int order);
int isInteger(char* string);

int main(int argc, char *argv[])
{

    if(argc < 2)
       printf("Insufficent arguments.\n");
    else
    {
       int len = argc - 1;
       int arr[MAX],n=0,order=1;
       int i=1;
       for(;i<len;++i)
       {
       // Validation function. Warning if not integer.
       if(isInteger(argv[i]))
             arr[n++] = atoi(argv[i]);
       else
             printf("warning: '%s' is not an integer. will be
ignored.\n",argv[i+1]);
       }

       // Warning if entire argument list is invalid.s
       if(n == 0 )
             printf("The input list is invalid. Please enter an integer
array.\n");
       else
       {
             pid_t pid = vfork();
             if(pid == 0)
             {
                   sortArray(arr,0,n/2,1);
                   exit(0);
             }
             else
                   sortArray(arr,n/2,n,0);

             i=0;
             printf("Resultant Array is:\n");
             for(;i<n;++i)
                   printf("%d ",arr[i]);
             puts("");
       }
    }
    return 0;
}

void sortArray(int arr[],int start,int end,int order)
{
    int temp;                      // For swap purposes
    // Initiate Loops for simple Bubble Sort.
    int i=start;
    int n = end - start;
    for(;i<end;++i)
    {
       int j=start;
       for(;j<start+end-i-1;++j)
       {
             if(((order == 1) && (arr[j]>arr[j+1])) || ((order == 0) &&
(arr[j]<arr[j+1])))
             {
                   // Swap arr[j] and arr[j+1]
                   temp = arr[j];
                   arr[j] = arr[j+1];
                   arr[j+1] = temp;
             }
       }
    }
}
```

```c
// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
       character = str[i];
       // In case of '-'' symbol, continue if not last character in string.
       if(character == 45 && i<strlen(str)-1)
             continue;
       // Else get the decimal value (-48) and check condition.
       character -= 48;
       if(character < 0 || character > 9)
             return 0;
    }
    return 1;
}
```

Output:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ gcc -o out 6.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 1 4 2 1 ] -2 3 8 7 9 0
warning: ']' is not an integer. will be ignored.
Resultant Array is:
1 1 2 4 9 8 7 3 -2
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 1 4 2 1 6 -2 3 8 7 9 0
Resultant Array is:
1 1 2 4 6 9 8 7 3 -2
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ 
```

7. Implement a multiprocessing version of binary search where the parent searches for the key in the first half and subsequent splits while the child searches in the other half of the array. By default u can implement a search for the first occurrence and later extend to support multiple occurrence (duplicated elements search as well)

Logic Used

In this program we are required to implement binary search of a key in a given array. We intend to search the first half of the array in the parent process and the second half in the child process.

Since the array could be unsorted, we also sort the respective halves in the respective processes. Once the arrays are sorted, we can do binary search on each half and get the position of the key (if exists in the array) else -1.
To extend the program to count multiple occurrences of the key, we do a linear search from the positions obtained from the binary search on either side. Since the arrays would be sorted by this instance, a linear search on either side of the position returned from binary search would give all occurrences of the key in the array. Here we keep track of only the *count* of occurrences and not the position. Since we maintain a common variable to track *count*, we use *vfork()*.
The input array is accepted through the command line and the validation function (isInteger()) ensures that the inputs are integers.

Code

```
#define MAX 1000

void sortArray(int arr[],int start,int end,int order);
int isInteger(char* string);
int binarySearch(int arr[], int start, int end, int x);

int main(int argc, char *argv[])
{

    if(argc < 2)
      printf("Insufficient arguments.\n");
    else
    {
      int len = argc - 2,x;
      if(isInteger(argv[argc -1]))
            x = atoi(argv[argc-1]);
      else
      {
            printf("Invalid search key provided.\n");
            return 0;
      }
      int arr[MAX],n=0,order=1;
      int i=1;
      for(;i<=len;++i)
```

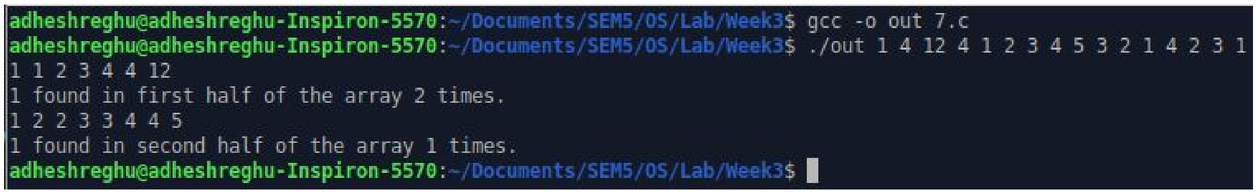
```
        {
        // Validation function. Warning if not integer.
        if(isInteger(argv[i]))
              arr[n++] = atoi(argv[i]);
        else
              printf("warning: '%s' is not an integer. will be
ignored.\n",argv[i]);
        }

        // Warning if entire argument list is invalid.s
        if(n == 0 )
              printf("The input list is invalid. Please enter an integer
array.\n");
        else
        {
              pid_t pid = vfork();

              if(pid == 0)
              {
                    sortArray(arr,0,n/2,1);
                    int pos = binarySearch(arr,0,(n/2) - 1,x);
                    if(pos!=-1)
                    {
                          int count = 1;
                          int j = pos+1;
                          if(pos+1 < n/2)
                          {
                                while((arr[j]==x) && (j<n/2))
                                {
                                      count = count +1;
                                      j = j+1;
                                }
                          }
                          j = pos-1;
                          if(pos-1 >= 0)
                          {
                                while((arr[j] == x) && (j>=0))
                                {
```

```
                        count = count + 1;
                        j = j-1;
                    }
                }
                printf("%d found in first half of the array %d
times.\n",x,count);
            }
            exit(0);
        }
        else
        {
            sortArray(arr,n/2,n,1);
            int pos = binarySearch(arr,n/2,n-1,x);
            if(pos!=-1)
            {
                int count = 1;
                int j = pos+1;
                if(pos+1 < n)
                {
                    while((arr[j]==x) && (j<n))
                    {
                        ++count;
                        ++j;
                    }
                }
                j = pos-1;
                if(pos-1 >= n/2)
                {
                    while((arr[j] == x) && (j>=n/2))
                    {
                        ++count;
                        --j;
                    }
                }
                printf("%d found in second half of the array %d
times.\n",x,count);
            }
        }
```

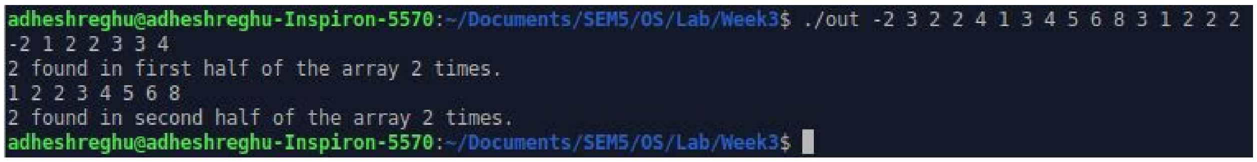
```
            wait(NULL);
        }
    }
    return 0;
}

void sortArray(int arr[],int start,int end,int order)
{
    int temp;
      // For swap purposes

    // Initiate Loops for simple Bubble Sort.
    int i=start;
    int n = end - start;
    for(;i<end;++i)
    {
      int j=start;
      for(;j<start+end-i-1;++j)
      {
            if(((order == 1) && (arr[j]>arr[j+1])) || ((order == 0) &&
(arr[j]<arr[j+1])))
            {
                  // Swap arr[j] and arr[j+1]
                  temp = arr[j];
                  arr[j] = arr[j+1];
                  arr[j+1] = temp;
            }
      }
    }

    i=start;
    for(;i<end;++i)
      printf("%d ",arr[i]);
    puts("");
}

// Validation Function
int isInteger(char* str)
```

```
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
      character = str[i];
      // In case of '-'' symbol, continue if not last character in string.
      if(character == 45 && i<strlen(str)-1)
           continue;
      // Else get the decimal value (-48) and check condition.
      character -= 48;
      if(character < 0 || character > 9)
           return 0;
    }
    return 1;
}

int binarySearch(int arr[], int start, int end, int x)
{
     if (end >= start)
     {
     int mid = start + (end - start) / 2;

     if (arr[mid] == x)
          return mid;

     if (arr[mid] > x)
          return binarySearch(arr, start, mid - 1, x);

     return binarySearch(arr, mid + 1, end, x);
     }

     return -1;
}
```

Output:

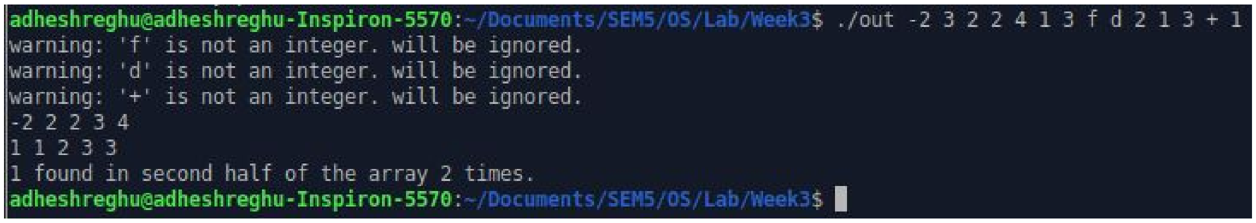


```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ gcc -o out 7.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 1 4 12 4 1 2 3 4 5 3 2 1 4 2 3 1
1 1 2 3 4 4 12
1 found in first half of the array 2 times.
1 2 2 3 3 4 4 5
1 found in second half of the array 1 times.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out -2 3 2 2 4 1 3 4 5 6 8 3 1 2 2 2
-2 1 2 2 3 3 4
2 found in first half of the array 2 times.
1 2 2 3 4 5 6 8
2 found in second half of the array 2 times.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$
```

With validation:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out -2 3 2 2 4 1 3 f d 2 1 3 + 1
warning: 'f' is not an integer. will be ignored.
warning: 'd' is not an integer. will be ignored.
warning: '+' is not an integer. will be ignored.
-2 2 2 3 4
1 1 2 3 3
1 found in second half of the array 2 times.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$
```

8. Read upon efficient ways of parallelizing the generation of Fibonacci series and apply the logic in a parent child relationship to contributes a faster version of fib series generation as opposed to sequential logic in (4)

Logic Used

In this program we are required to parallelize the generation of Fibonacci series using fork. Let us take a look at the general algorithms available to us:

- Bottom-up Approach - In this algorithm we generate the fibonacci series in a bottom up fashion and store the previous fibonacci terms in an array, thus reducing the complexity to recompute terms. This is a DP Approach. It takes O(n) time complexity. However we cannot parallelize this algorithm as there are data dependencies. A term cannot be computed without computing the previous two terms. If we think of computing the previous two terms parallely, it breaks the definition of the algorithm. Hence we cannot parallelize this algorithm.

- Recursive Approach - In this algorithm we compute Fib(n) by computing Fib(n-1) and Fib(n-2) recursively. A standard definition of the algorithm is

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

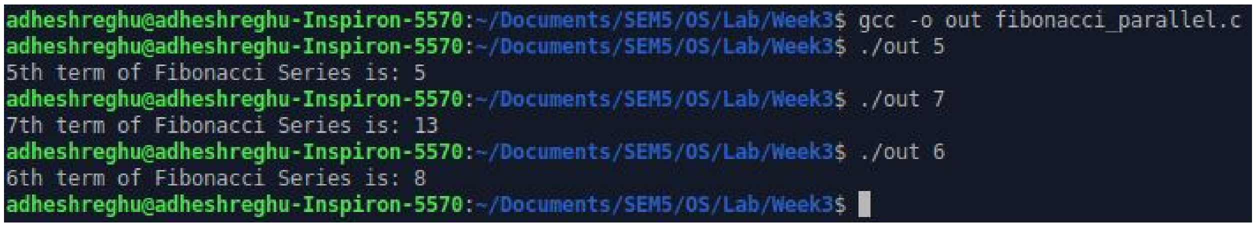
This algorithm takes time complexity of O(2^n). This algorithm can be easily parallelized. We can compute the left, i.e., Fib(n-1), and right, i.e., Fib(n-2) parallely in two different processes. This could help speed up the process. Since we have to return the sum of the results from the two processes, we have to share data and hence we use vfork(). We wait for both the processes to complete execution and then we return the sum. This would help speed up the calculation.

Code

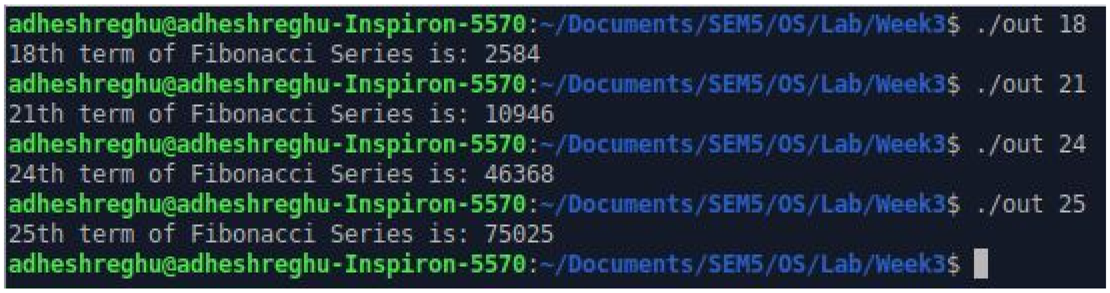
```
long int fibonacci(int  n);

int main(int argc, char *argv[])
{
    if(argc < 2)
       printf("Insufficient arguments\n");
    else if(argc == 2)
    {
       int n = atoi(argv[1]);
       long int fib = fibonacci(n);
       printf("%dth term of Fibonacci Series is: %ld\n",n,fib);
    }
    else
       printf("Too many arguments.\n");
    return 0;
}
```

```c
long int fibonacci(int  n)
{
    if(n<=2)
       return 1;
    else
    {
       pid_t pid = vfork();
       long int fib_0,fib_1;
       if(pid == 0)
       {
             fib_0 = fibonacci(n-2);
             exit(0);
       }
       else if(pid > 0)
       {
             fib_1 = fibonacci(n-1);
       }
       wait(NULL);
       return fib_0 + fib_1;
    }
}
```

Output



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ gcc -o out fibonacci_parallel.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 5
5th term of Fibonacci Series is: 5
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 7
7th term of Fibonacci Series is: 13
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 6
6th term of Fibonacci Series is: 8
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 18
18th term of Fibonacci Series is: 2584
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 21
21th term of Fibonacci Series is: 10946
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 24
24th term of Fibonacci Series is: 46368
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$ ./out 25
25th term of Fibonacci Series is: 75025
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week3$
```