

OPERATING SYSTEMS - COM301P

LAB ASSIGNMENT - 3

FORKING ASSIGNMENT - 2

VIKNESH RAJARAMON

COE18B060

(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:

- Get the number of terms required in both the series from the user. After that call **fork()**.
- Generate the Odd Series in Parent process by taking the first element in the array as '1' and in each iteration, increase the value by 2.
- Generate the Even Series in Child process by taking the first element in the array as '2' and in each iteration, increase the value by 2.

C CODE:

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

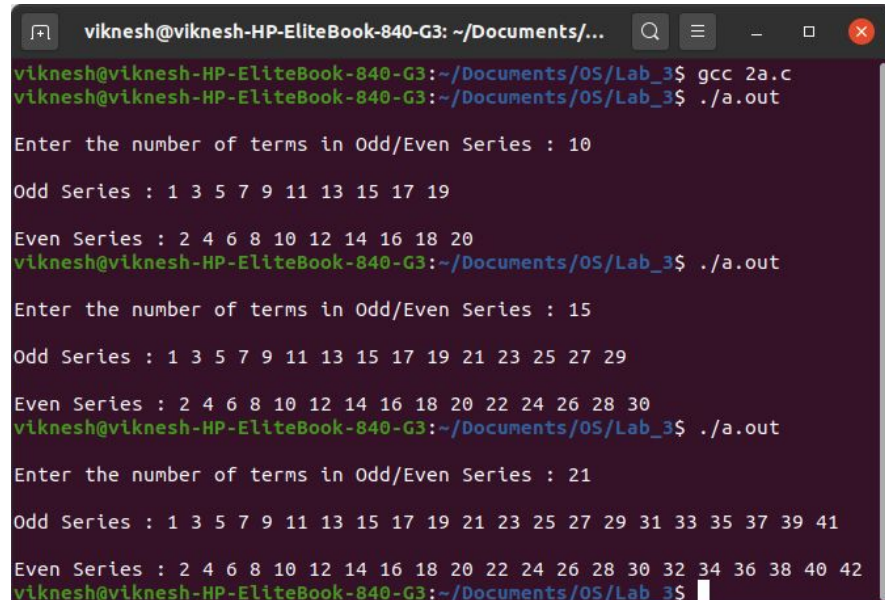
int main()
{
    int n;
    printf("\nEnter the number of terms in Odd/Even Series : ");
    scanf("%d",&n);
    int array[n];
    pid_t pid;
    pid=fork();
    if(pid>0)
    {
        int start=1;
        for(int i=0;i<n;++i)
        {
            array[i]=start;
            start+=2;
        }
        printf("\nOdd ");
    }
    else if(pid==0)
    {
        int start=2;
        for(int i=0;i<n;++i)
        {
            array[i]=start;
            start+=2;
        }
        printf("\nEven ");
    }
    else
    {
        printf("\nForking failed...\n");
    }
    printf("Series : ");
    for(int i=0;i<n;++i)
    {
        printf("%d ",array[i]);
    }
}
```

```

    }
    printf("\n");
    return 0;
}

```

OUTPUT:



```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 2a.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Odd/Even Series : 10

Odd Series : 1 3 5 7 9 11 13 15 17 19

Even Series : 2 4 6 8 10 12 14 16 18 20
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Odd/Even Series : 15

Odd Series : 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29

Even Series : 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Odd/Even Series : 21

Odd Series : 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41

Even Series : 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$

```

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

LOGIC:

- Get the number of terms in the series. After that call **fork()**.
- In the Parent process, scan the array and check if the number is not divisible by 2. If yes, then add the number to sum. If not, ignore the value.
- In the Child process, scan the array and check if the number is divisible by 2. If yes, then add the number to sum. If not, ignore the value.

C CODE:

```

#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include<time.h>

int main()
{
    srand(time(0));
    int n;
    printf("\nEnter the number of terms in Odd/Even Series : ");
    scanf("%d",&n);
    int array[n];
    printf("\nArray Elements : ");
    for(int i=0;i<n;++i)
    {
        array[i]=rand()%1000+1;
        printf("%d ",array[i]);
    }
}

```

```

}
printf("\n");
pid_t pid;
pid=fork();
int sum=0;
if(pid>0)
{
    printf("\nOdd Series Elements : ");
    for(int i=0;i<n;++i)
    {
        if(array[i]%2==1)
        {
            printf("%d ",array[i]);
            sum+=array[i];
        }
    }
    printf("\nOdd ");
}
else if(pid==0)
{
    printf("\nEven Series Elements : ");
    for(int i=0;i<n;++i)
    {
        if(array[i]%2==0)
        {
            printf("%d ",array[i]);
            sum+=array[i];
        }
    }
    printf("\nEven ");
}
else
{
    printf("\nForking failed...\n");
}
printf("Series Sum : %d\n",sum);
return 0;
}

```

OUTPUT:

```
viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 2b.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Odd/Even Series : 10

Array Elements : 384 887 778 916 794 336 387 493 650 422

Odd Series Elements : 887 387 493
Odd Series Sum : 1767

Even Series Elements : 384 778 916 794 336 650 422
Even Series Sum : 4280
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Odd/Even Series : 15

Array Elements : 384 887 778 916 794 336 387 493 650 422 363 28 691 60 764

Odd Series Elements : 887 387 493 363 691
Odd Series Sum : 2821

Even Series Elements : 384 778 916 794 336 650 422 28 60 764
Even Series Sum : 5132
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$
```

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

LOGIC:

- Get the lower bound and upper bound of the range between which we need to list the Armstrong numbers.
- Start a loop from lower bound to upper bound of the range.
- Inside the loop, we use **vfork()**. **vfork()** shares the same memory between Parent process and Child process.
- Inside the Child process, calculate the number of digits and find the sum of each digit raised to the power of the number of digits.
- Inside the Parent process, check whether the sum and the number are equal. If yes, then print the number.

C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<math.h>

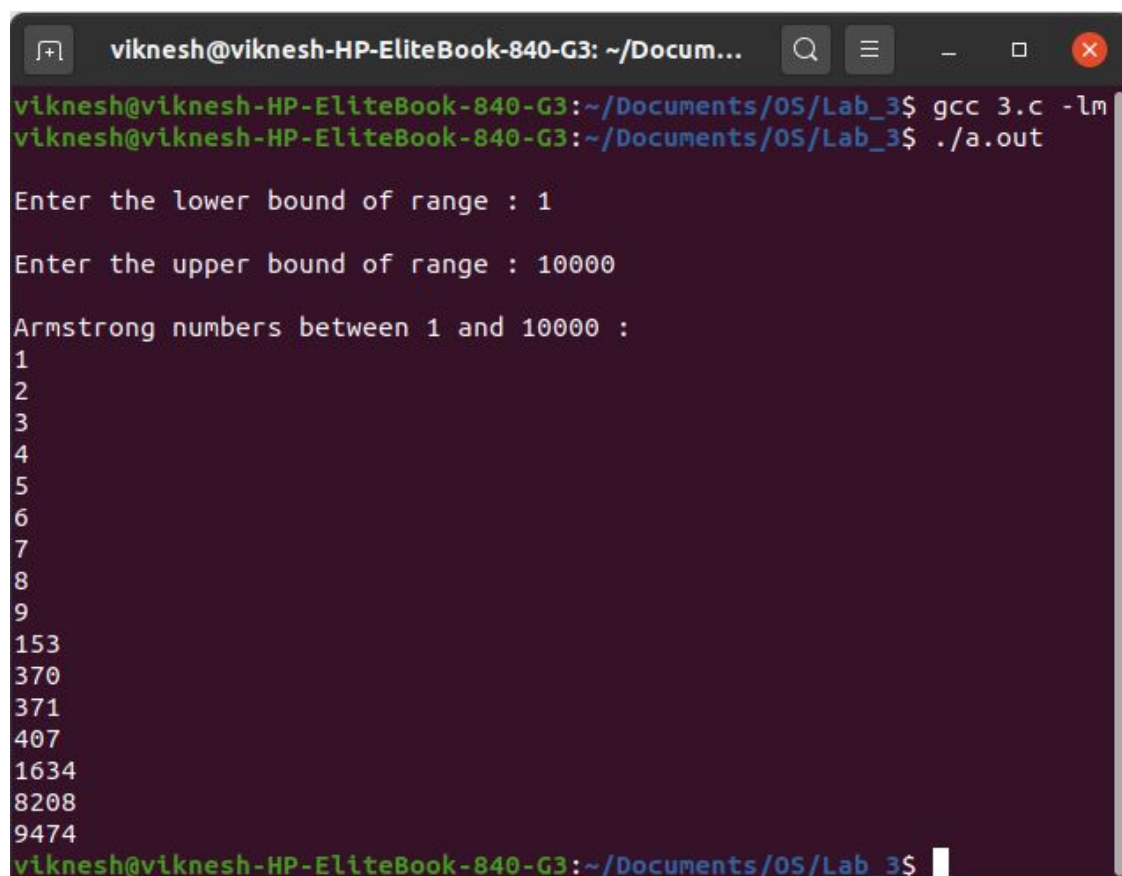
int main()
{
    pid_t pid;
    int a,b;
    int sum=0,x=0;
    int digits=0;
    printf("\nEnter the lower bound of range : ");
    scanf("%d",&a);
    printf("\nEnter the upper bound of range : ");
    scanf("%d",&b);
```

```

printf("\nArmstrong numbers between %d and %d : \n",a,b);
for(int i=a;i<=b;++i)
{
    pid=vfork();
    if(pid==0)
    {
        x=i;
        sum=0;
        digits=(int) log10(x)+1;
        while(x)
        {
            int r=x%10;
            x/=10;
            sum+=pow(r,digits);
        }
        exit(0);
    }
    else if(pid>0)
    {
        if(sum==i)
        {
            printf("%d\n",i);
        }
    }
    else
    {
        printf("\nForking error...\n");
    }
}
return 0;
}

```

OUTPUT:



The screenshot shows a terminal window with the following content:

```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Docum...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 3.c -lm
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the lower bound of range : 1
Enter the upper bound of range : 10000

Armstrong numbers between 1 and 10000 :
1
2
3
4
5
6
7
8
9
153
370
371
407
1634
8208
9474
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$

```

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

LOGIC:

- Get the number of terms in Fibonacci/Prime series. After that call **fork()**.
- In the Parent process, run the code for Prime number checking. If it is a prime number, store the number in an array.
- Inside the Child process, run the code for Fibonacci number using DP approach.
- Finally print both the series separately.

C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

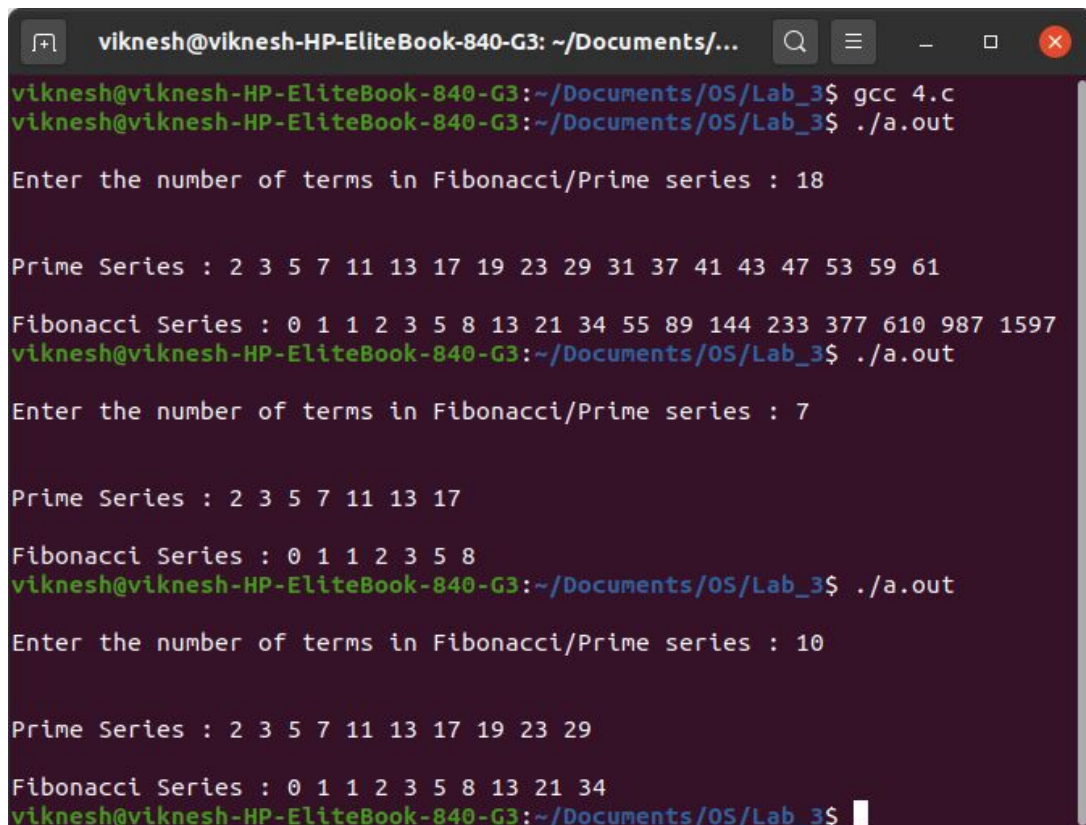
int main()
{
    int n;
    printf("\nEnter the number of terms in Fibonacci/Prime series : ");
    scanf("%d",&n);
    printf("\n");
    unsigned long int array[n];
    pid_t pid=fork();
    if(pid>0)
    {
        int count=2;
        array[0]=2;
        array[1]=3;
        for(int i=5;count<=n;++i)
        {
            int flag=0;
            if((i%2==0) || (i%3==0))
            {
                flag=1;
            }
            else
            {
                for(int j=5;j*j<=i;j=j+6)
                {
                    if((i%j==0) || (i%(j+2)==0))
                    {
                        flag=1;
                        break;
                    }
                }
            }
            if(flag==0)
            {
                array[count]=i;
                ++count;
            }
        }
    }
}
```

```

    }
}
else if(pid==0)
{
    array[0]=0;
    array[1]=1;
    for(int i=2;i<n;++i)
    {
        array[i]=array[i-1]+array[i-2];
    }
}
else
{
    printf("\nForking failed...\n");
}
if(pid==0)
{
    printf("\nFibonacci");
}
else if(pid>0)
{
    printf("\nPrime");
}
printf(" Series : ");
for(int i=0;i<n;++i)
{
    printf("%lu ",array[i]);
}
printf("\n");
return 0;
}

```

OUTPUT:



```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 4.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Fibonacci/Prime series : 18

Prime Series : 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61
Fibonacci Series : 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Fibonacci/Prime series : 7

Prime Series : 2 3 5 7 11 13 17
Fibonacci Series : 0 1 1 2 3 5 8
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Fibonacci/Prime series : 10

Prime Series : 2 3 5 7 11 13 17 19 23 29
Fibonacci Series : 0 1 1 2 3 5 8 13 21 34
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$

```


(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:

- Get the number of elements in the array. Generate the elements of the array and print them. After that call **fork()**.
- In the Child process, run the **qsort()** to sort the array in ascending order.
- In the Parent process, run the **qsort()** to sort the array in descending order.
- Since **fork()** creates separate memory locations for parent and child processes, both the arrays will be stored separately.
- Finally print both the arrays.

C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include<time.h>

int ascending(const void *a,const void *b)
{
    return (*(int *)a - *(int *)b);
}

int descending(const void *a,const void *b)
{
    return (*(int *)b - *(int *)a);
}

int main()
{
    srand(time(0));
    int n;
    printf("\nEnter the number of elements : ");
    scanf("%d",&n);
    int array[n];
    printf("\nArray elements : ");
    for(int i=0;i<n;++i)
    {
        array[i]=rand()%1000 + 1;
        printf("%d ",array[i]);
    }
    printf("\n");
    pid_t pid=fork();
    if(pid==0)
    {
        qsort(array,n,sizeof(int),ascending);
        printf("\nAscending : ");
    }
    else if(pid>0)
```

```

{
    qsort(array,n,sizeof(int),descending);
    printf("\nDescending : ");
}
else
{
    printf("\nForking failed...\n");
}
for(int i=0;i<n;++i)
{
    printf("%d ",array[i]);
}
printf("\n");
return 0;
}

```

OUTPUT:

```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_3
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 5.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of elements : 10

Array elements : 862 584 860 769 921 651 545 450 191 297

Descending : 921 862 860 769 651 584 545 450 297 191

Ascending : 191 297 450 545 584 651 769 860 862 921
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of elements : 18

Array elements : 150 310 603 747 88 59 52 302 736 312 77 4 889 320 939 916 889 571

Descending : 939 916 889 889 747 736 603 571 320 312 310 302 150 88 77 59 52 4

Ascending : 4 52 59 77 88 150 302 310 312 320 571 603 736 747 889 889 916 939
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$

```

(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:

- Get the number of elements in the array. Generate the elements of the array and print them.
- Divide the array into two halves. If the size of the array is 'n' and $n\%2==0$, then the size of each array is $n/2$. If $n\%2!=0$, then size of the first half is $n/2$ and size of the second half is $“(n/2)+1”$.
- Call **vfork()**.
- Since **vfork()** shares the same memory between parent and child processes, both the arrays will be stored together.
- In the Child process, run the **qsort()** to sort the first half ascending order.
- In the Parent process, run the **qsort()** to sort the second half descending order.
- Finally print both the arrays.

C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include<time.h>

int ascending(const void *a,const void *b)
{
    return (*(int *)a - *(int *)b);
}

int descending(const void *a,const void *b)
{
    return (*(int *)b - *(int *)a);
}

int main()
{
    srand(time(0));
    int n;
    printf("\nEnter the number of elements : ");
    scanf("%d",&n);
    int array[n];
    printf("\nArray elements : ");
    for(int i=0;i<n;++i)
    {
        array[i]=rand()%1000 + 1;
        printf("%d ",array[i]);
    }
    printf("\n");
    int size1=n/2;
    int size2=n-size1;
    pid_t pid=vfork();
    if(pid==0)
    {
        qsort(&(array[0]),size1,sizeof(int),ascending);
        exit(0);
    }
    else if(pid>0)
    {
        qsort(&(array[size1]),size2,sizeof(int),descending);
    }
    else
    {
        printf("\nForking failed...\n");
    }
    printf("\nArray : ");
    for(int i=0;i<n;++i)
    {
        printf("%d ",array[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT:

```
viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_3
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 6.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of elements : 10

Array elements : 557 142 335 392 539 245 549 318 944 486

Array : 142 335 392 539 557 944 549 486 318 245
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 6.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of elements : 9

Array elements : 776 530 448 791 547 318 29 588 43

Array : 448 530 776 791 588 547 318 43 29
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$
```

(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)

First occurrence

LOGIC:

- Get the number of elements in the array. Generate the elements of the array and print them.
- Divide the array into two halves. If the size of the array is 'n' and $n\%2==0$, then the size of each array is $n/2$. If $n\%2!=0$, then the size of the first half is $n/2$ and size of the second half is $(n/2)+1$.
- Get the element to be searched for in the array and call **fork()**.
- Since **fork()** creates different memory locations for Child and Parent processes, the arrays will be stored separately.
- In the Parent process, run the **qsort()** to sort the first half in ascending order and run **Binary search** to find the first occurrence of the search element in the first half.
- In the Child process, run the **qsort()** to sort the second half in ascending order and run **Binary search** to find the first occurrence of the search element in the second half.
- Print the position of the search element in both the first half and second half separately.

C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include<time.h>

int sort(const void *a,const void *b)
{
    return (*(int *)a - *(int *)b);
}

int binary_search(int array[],int lower,int upper,int search_element)
```

```

{
    if(upper>=lower)
    {
        int mid=lower+(upper-lower)/2;
        if(array[mid]==search_element)
        {
            return mid;
        }
        if(array[mid]>search_element)
        {
            return binary_search(array,lower,mid-1,search_element);
        }
        return binary_search(array,mid+1,upper,search_element);
    }
    return -1;
}

int main()
{
    srand(time(0));
    int n,x;
    printf("\nEnter the number of elements : ");
    scanf("%d",&n);
    int array[n];
    printf("\nArray elements : ");
    for(int i=0;i<n;++i)
    {
        array[i]=rand()%1000+1;
        printf("%d ",array[i]);
    }
    printf("\nEnter the element to search for : ");
    scanf("%d",&x);
    printf("\n");
    int size=n/2;
    int pos;
    pid_t pid;
    pid=fork();
    if(pid>0)
    {
        qsort(&(array[0]),size,sizeof(int),sort);
        printf("\nSorted Array (First Half) : ");
        for(int i=0;i<size;++i)
        {
            printf("%d ",array[i]);
        }
        printf("\n");
        pos=binary_search(array,0,size-1,x);
        printf("First Half Position : ");
    }
    else if(pid==0)
    {
        qsort(&(array[size]),n-size,sizeof(int),sort);
        printf("\nSorted Array (Second Half) : ");
        for(int i=size;i<n;++i)
        {
            printf("%d ",array[i]);
        }
    }
}

```

```

        printf("\n");
        pos=binary_search(array,size,n-1,x);
        pos-=size;
        printf("Second Half Position : ");
    }
    else
    {
        printf("\nforking Failed...\n");
    }
    if(pos<0)
    {
        printf("%d not found\n",x);
    }
    else
    {
        printf("%d\n",pos);
    }
    return 0;
}

```

OUTPUT:

```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_3
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 7a.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of elements : 15

Array elements : 784 747 557 247 313 934 220 652 500 789 817 492 816 253 302
Enter the element to search for : 253

Sorted Array (First Half) : 220 247 313 557 747 784 934
First Half Position : 253 not found

Sorted Array (Second Half) : 253 302 492 500 652 789 816 817
Second Half Position : 0
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of elements : 16

Array elements : 152 730 254 752 381 524 419 159 674 214 872 826 217 401 700 763
Enter the element to search for : 152

Sorted Array (First Half) : 152 159 254 381 419 524 730 752
First Half Position : 0

Sorted Array (Second Half) : 214 217 401 674 700 763 826 872
Second Half Position : 152 not found
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$

```

Multiple occurrence

LOGIC:

- Get the number of elements in the array. Generate the elements of the array and print them.
- Divide the array into two halves. If the size of the array is 'n' and $n\%2==0$, then the size of each array is $n/2$. If $n\%2!=0$, then the size of the first half is $n/2$ and the size of the second

half is " $(n/2)+1$ ".

- Get the element to be searched for in the array and call **fork()**.
- Since **fork()** creates different memory locations for Child and Parent processes, the arrays will be stored separately.
- In the Parent process, run the **qsort()** to sort the first half in ascending order. After sorting, call functions **first_occurrence** and **last_occurrence** to find the first occurrence and last occurrence of the search element in the first half array.
- In the Child process, run the **qsort()** to sort the second half in ascending order. After sorting, call functions **first_occurrence** and **last_occurrence** to find the first occurrence and last occurrence of the search element in the second half array.
- Print the number of occurrences of the search element in the first half and second half separately.

C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<time.h>

int sort(const void *a,const void *b)
{
    return (*(int *)a - *(int *)b);
}

int first_occurrence(int array[],int lower,int upper,int x)
{
    if(lower>upper)
    {
        return lower;
    }
    int mid=lower+(upper-lower)/2;
    if(array[mid]>=x)
    {
        return first_occurrence(array,lower,mid-1,x);
    }
    return first_occurrence(array,mid+1,upper,x);
}

int last_occurrence(int array[],int lower,int upper,int x)
{
    if(lower>upper)
    {
        return lower;
    }
    int mid=lower+(upper-lower)/2;
    if(array[mid]>x)
    {
        return last_occurrence(array,lower,mid-1,x);
    }
    return last_occurrence(array,mid+1,upper,x);
}
```

```

}

int main()
{
    srand(time(0));
    int n,x;
    printf("\nEnter the number of elements : ");
    scanf("%d",&n);
    int array[n];
    char string[7];
    printf("\nUnsorted Array elements : " );
    for(int i=0;i<n;++i)
    {
        array[i]=rand()%10+1;
        printf("%d ",array[i]);
    }
    printf("\nEnter the element to search for : ");
    scanf("%d",&x);
    printf("\n");
    int start=0,end=n;
    int first_pos=0,last_pos=0;
    pid_t pid;
    pid=fork();
    if(pid>0)
    {
        qsort(&(array[0]),n/2,sizeof(int),sort);
        first_pos=first_occurrence(array,0,n/2-1,x);
        last_pos=last_occurrence(array,0,n/2-1,x);
        strcpy(string,"First");
        end=n/2;
    }
    else if(pid==0)
    {
        qsort(&(array[n/2]),n-n/2,sizeof(int),sort);
        first_pos=first_occurrence(array,n/2,n-1,x);
        last_pos=last_occurrence(array,n/2,n-1,x);
        first_pos-=n/2;
        last_pos-=n/2;
        strcpy(string,"Second");
        start=n/2;
    }
    else
    {
        printf("\nforking Failed...\n");
    }
    printf("\nArray elements in %s Half : ",string);
    for(;start<end;++start)
    {
        printf("%d ",array[start]);
    }
    printf("\nNumber of occurrences in %s Half : %d\n",string,last_pos-first_pos);
    return 0;
}

```


OUTPUT:

```
viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_3
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 7b.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of elements : 14

Unsorted Array elements : 1 7 8 10 8 1 2 5 9 8 2 5 9 8

Enter the element to search for : 8

Array elements in First Half : 1 1 2 7 8 8 10
Number of occurrences in First Half : 2

Array elements in Second Half : 2 5 5 8 8 9 9
Number of occurrences in Second Half : 2
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of elements : 9

Unsorted Array elements : 4 10 8 3 9 9 9 6 6

Enter the element to search for : 9

Array elements in First Half : 3 4 8 10
Number of occurrences in First Half : 0

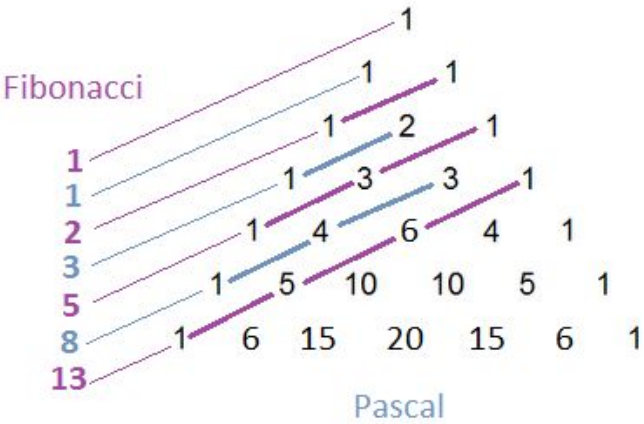
Array elements in Second Half : 6 6 9 9 9
Number of occurrences in Second Half : 3
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$
```

(8) * Non Mandatory [extra credits]

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:

- Get the number of terms in the Fibonacci series. Assume it to be 'n'.
- Calculate Pascal's triangle upto line 'n' and store it in an array.
- For odd position (say **pos_odd**) terms in the series, find the sum of terms in the Pascal's triangle (say **array[i][j]**) such that for two values i,j where **i<j, i+j=pos_odd**.
- For even position (say **pos_even**) terms in the series, find the sum of terms in the Pascal's triangle (say **array[i][j]**) such that for two values i,j where **i<j, i+j=pos_even**.
- Finally, print the series.



C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<string.h>
#include<unistd.h>

void pascal_triangle(unsigned long int array[], int n)
{
    unsigned long int c=1;
    for(unsigned long int i=1;i<=n;++i)
    {
        array[i-1]=c;
        c=c*(n-i)/i;
    }
}

void fibonacci(unsigned long int fib[],int n)
{
    pid_t pid;
    unsigned long int array[n][n];
    for(int i=1;i<=n;++i)
    {
        unsigned long int a[i];
        memset(a,0,i);
        pascal_triangle(a,i);
        for(int j=0;j<i;++j)
        {
            array[i-1][j]=a[j];
        }
    }
    pid=vfork();
    if(pid==0)
    {
        for(int i=0;i<n;i+=2)
        {
            fib[i]=0;
            for(int j=0;j<=i/2;++j)
            {
                fib[i]+=array[i-j][j];
            }
        }
        exit(0);
    }
    else if(pid>0)
    {
        for(int i=1;i<n;i+=2)
        {
            fib[i]=0;
            for(int j=0;j<=i/2;++j)
            {
                fib[i]+=array[i-j][j];
            }
        }
    }
    else
```

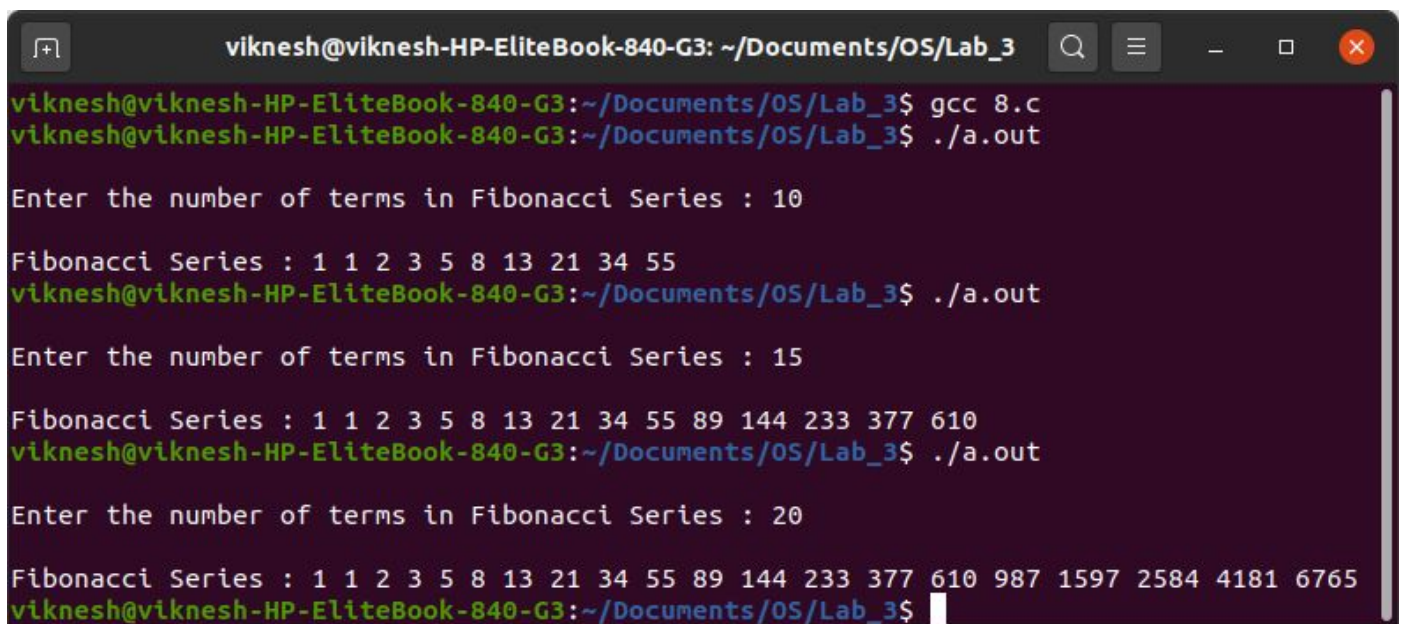
```

    {
        printf("\nForking Failed...\n");
    }
}

int main()
{
    int n;
    printf("\nEnter the number of terms in Fibonacci Series : ");
    scanf("%d",&n);
    unsigned long int fib[n];
    fibonacci(fib,n);
    printf("\nFibonacci Series : ");
    for(int i=0;i<n;++i)
    {
        printf("%lu ",fib[i]);
    }
    printf("\n");
    return 0;
}

```

OUTPUT:



The screenshot shows a terminal window with the following content:

```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_3
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ gcc 8.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Fibonacci Series : 10

Fibonacci Series : 1 1 2 3 5 8 13 21 34 55
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Fibonacci Series : 15

Fibonacci Series : 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$ ./a.out

Enter the number of terms in Fibonacci Series : 20

Fibonacci Series : 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_3$

```