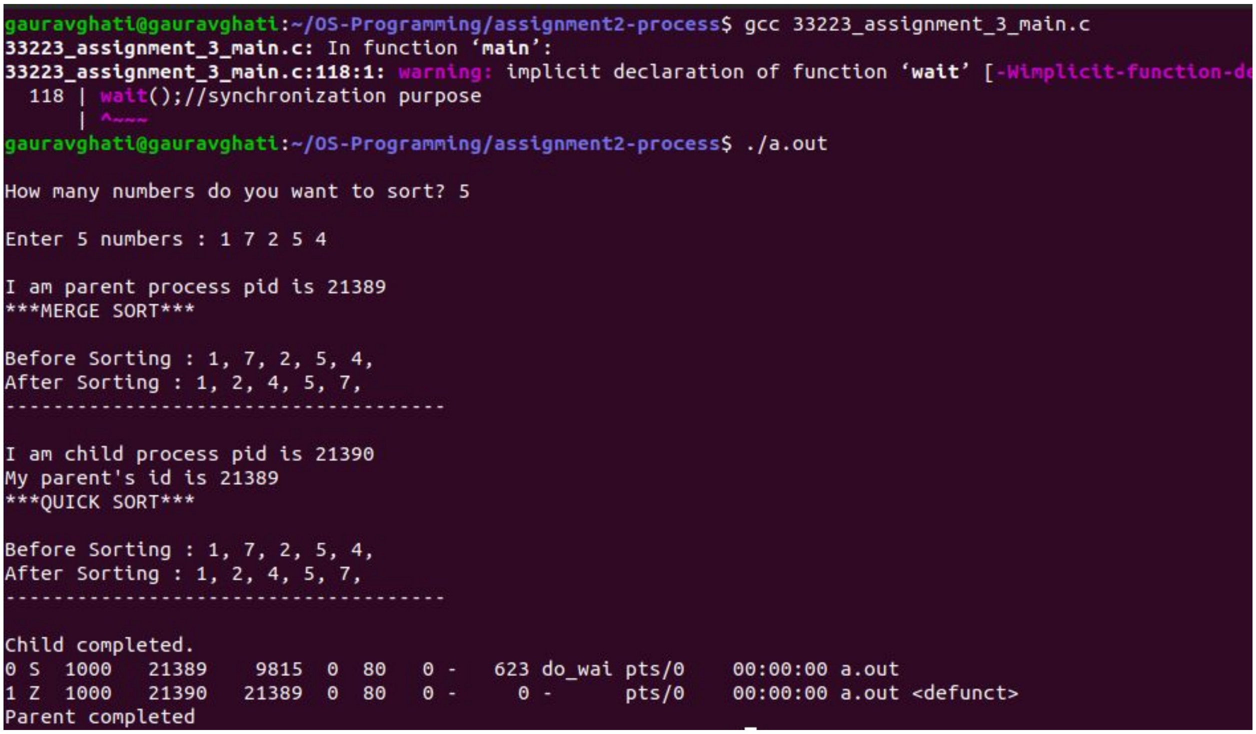
Assignment 2

```
/* Name-Gaurav Ghati
Div- TE10
Batch-L10
Rollno-33223

Part A)
Implement the C program in which main program accepts the integers to be sorted. Main
program uses the FORK system call to create a new process called a child
process. Parent process sorts the integers using sorting algorithm and waits for child
process using WAIT system call to sort the integers using any sorting algorithm. Also
demonstrate zombie and orphan states.*/

#include<stdio.h>
#include<sys/types.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<ctype.h>
#define MAX 20
int partition(int arr[], int beg, int end) {
int pivot = arr[beg], i = beg, j = end + 1, temp;
do {
do
i++;
while(arr[i] < pivot && i <= end);
do
j--;
while(arr[j] > pivot);
if(i < j) {
temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
}
}while(i < j);
arr[beg] = arr[j];
arr[j] = pivot;
return j;
}
void merge(int a[], int beg, int end, int mid) {
int res[10], i = beg, j = mid + 1, k = 0;
```

```
while(i <= mid && j <= end) {
if(a[i] < a[j]) {
res[k] = a[i];
k++, i++;
}

else {
res[k] = a[j];
k++, j++;
}
}
while(i <= mid) {
res[k] = a[i];
k++, i++;
}
while(j <= end) {
res[k] = a[j];
k++, j++;
}
for(i = beg, j = 0; i <= end; i++, j++)
a[i] = res[j];
}
void merge_sort(int arr[], int beg, int end) {
int mid;
if(beg < end) {
mid = (beg + end) / 2;
merge_sort(arr, beg, mid);
merge_sort(arr, mid + 1, end);
merge(arr, beg, end, mid);
}
}
void quick_sort(int arr[], int beg, int end) {
int piv_index;
if(beg < end) {
piv_index = partition(arr, beg, end);
quick_sort(arr, beg, piv_index - 1);
quick_sort(arr, piv_index + 1, end);
}
}
int main(void)
{
int pid, n, arr[MAX];
printf("\nHow many numbers do you want to sort? ");
scanf("%d", &n);
```

```
printf("\nEnter %d numbers : ", n);
for(int i = 0; i < n; i++)
scanf("%d", &arr[i]);
pid = fork();
switch(pid) {
case 0:
printf("\nI am child process pid is %d", getpid());
printf("\nMy parent's id is %d", getppid());
printf("\n***QUICK SORT***\n");
printf("\nBefore Sorting : ");
for(int i = 0; i < n; i++)
printf("%d, ", arr[i]);
quick_sort(arr, 0, n-1);
printf("\nAfter Sorting : ");
for(int i = 0; i < n; i++)

printf("%d, ", arr[i]);
printf("\n------------------------------------\n");
printf("\nChild completed.\n");
//system("ps -al");
break;
case -1:
printf("\nError");
default:
//sleep(10);
printf("\nI am parent process pid is %d", getpid());
printf("\n***MERGE SORT***\n");
printf("\nBefore Sorting : ");
for(int i = 0; i < n; i++)
printf("%d, ", arr[i]);
merge_sort(arr, 0, n-1);
printf("\nAfter Sorting : ");
for(int i = 0; i < n; i++)
printf("%d, ", arr[i]);
printf("\n------------------------------------\n");
system("ps -al | grep a.out");
wait();//synchronization purpose
printf("Parent completed\n");
}
//system("ps -ax");
return 0;
}
```

OUTPUT PART A



```
gauravghati@gauravghati:~/OS-Programming/assignment2-process$ gcc 33223_assignment_3_main.c
33223_assignment_3_main.c: In function ‘main’:
33223_assignment_3_main.c:118:1: warning: implicit declaration of function ‘wait’ [-Wimplicit-function-de
  118 | wait();//synchronization purpose
      | ^~~~
gauravghati@gauravghati:~/OS-Programming/assignment2-process$ ./a.out

How many numbers do you want to sort? 5

Enter 5 numbers : 1 7 2 5 4

I am parent process pid is 21389
***MERGE SORT***

Before Sorting : 1, 7, 2, 5, 4,
After Sorting : 1, 2, 4, 5, 7,
-------------------------------------

I am child process pid is 21390
My parent's id is 21389
***QUICK SORT***

Before Sorting : 1, 7, 2, 5, 4,
After Sorting : 1, 2, 4, 5, 7,
-------------------------------------

Child completed.
0 S  1000   21389    9815  0  80   0 -   623 do_wai pts/0    00:00:00 a.out
1 Z  1000   21390   21389  0  80   0 -     0 -      pts/0    00:00:00 a.out <defunct>
Parent completed
```

PART B)

/* Name-Gaurav Ghati
Div- TE10
Batch-L10
Rollno-33223

Part B)
Implement the C program in which main program accepts an integer array. Main program uses the
FORK system call to create a new process called a child process. Parent process sorts an integer array
and passes the sorted array to child process through the command line arguments of EXECVE system
call. The child process uses EXECVE system call to load new program that uses this sorted array for
performing the binary search to search the particular item in the array.*/

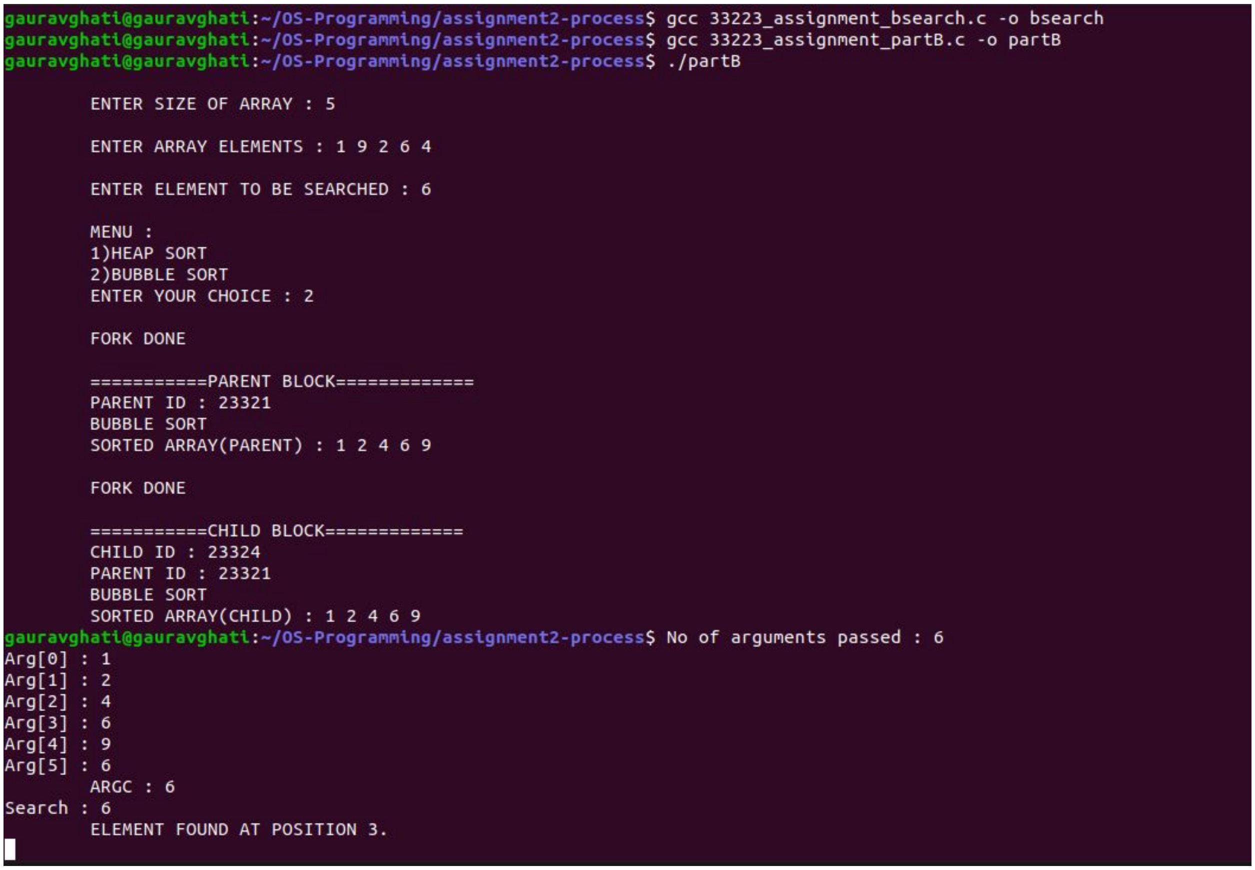
```
#include<stdio.h>
#include<sys/types.h>
#include<string.h>
#include<pthread.h>
```

```
#include<stdlib.h>
#include<unistd.h>
#include<ctype.h>
//#include <cstdlib>
#define MAX 20
// Print an array
void printArray(int arr[], int n)
{
for (int i = 0; i < n; ++i)
printf("%d ", arr[i]);
printf("\n");
}
void swap(int *a, int *b)
{
int t;
t=*a; *a=*b; *b=t;
}
void bubbleSort(int arr[], int n)
{
int i, j;
for (i = 0; i < n-1; i++)
for (j = 0; j < n-i-1; j++)
if (arr[j] > arr[j+1])
swap(&arr[j], &arr[j+1]);
}
void heapify(int arr[], int n, int i)
{
int largest = i; // Initialize largest as root
int l = 2*i + 1; // left = 2*i + 1
int r = 2*i + 2; // right = 2*i + 2

// If left child is larger than root
if (l < n && arr[l] > arr[largest])
largest = l;
// If right child is larger than largest so far
if (r < n && arr[r] > arr[largest])
largest = r;
// If largest is not root
if (largest != i)
{
swap(&arr[i], &arr[largest]);
// Recursively heapify the affected sub-tree
heapify(arr, n, largest);
}
```

```
}
void heapSort(int arr[], int n)
{
// Build heap (rearrange array)
for (int i = n / 2 - 1; i >= 0; i--)
heapify(arr, n, i);
// One by one extract an element from heap
for (int i=n-1; i>0; i--)
{
// Move current root to end
swap(&arr[0], &arr[i]);
// call max heapify on the reduced heap
heapify(arr, i, 0);
}
}

int main()
{
int process_id, size, array[30], i, choice, search, temp[20];
char str[30];
char *arg[30];
printf("\n\tENTER SIZE OF ARRAY : ");
scanf("%d", &size);
printf("\n\tENTER ARRAY ELEMENTS : ");
for(i=0; i<size; i++)
scanf("%d", &array[i]);
printf("\n\tENTER ELEMENT TO BE SEARCHED : ");
scanf("%d", &search);
//bubbleSort(array, size);
printf("\n\tMENU : \n\t1)HEAP SORT\n\t2)BUBBLE SORT\n\tENTER YOUR CHOICE : ");
scanf("%d", &choice);
process_id = fork();

printf("\n\tFORK DONE");
switch(process_id) {
case -1 : printf("\n\tERROR!");
break;

case 0 : printf("\n\n\t===========CHILD BLOCK=============");
printf("\n\tCHILD ID : %d", getpid());
printf("\n\tPARENT ID : %d", getppid());
switch(choice) {
case 1 : printf("\n\tHEAP SORT");
heapSort(array, size);
```

```
break;
case 2 : printf("\n\tBUBBLE SORT");
bubbleSort(array, size);
break;

}
printf("\n\tSORTED ARRAY(CHILD) : ");
printArray(array, size);
for(i=0; i<size; i++)
temp[i] = array[i];
temp[i] = search;
for(i=0; i<size+1; i++) {
sprintf(str, "%d", temp[i]);
arg[i] = malloc(sizeof(str));
strcpy(arg[i], str);
}
arg[i]=NULL;
execve("./BinarySearch.out", arg, NULL);
break;
default : printf("\n\n\t===========PARENT BLOCK============");
printf("\n\tPARENT ID : %d", getpid());

switch(choice) {
case 1 : printf("\n\tHEAP SORT");
heapSort(array, size);
break;
case 2 : printf("\n\tBUBBLE SORT");
bubbleSort(array, size);
break;

}
printf("\n\tSORTED ARRAY(PARENT) : ");
printArray(array, size);

break;

}

return 0;
}
```

OUTPUT B)



```
gauravghati@gauravghati:~/OS-Programming/assignment2-process$ gcc 33223_assignment_bsearch.c -o bsearch
gauravghati@gauravghati:~/OS-Programming/assignment2-process$ gcc 33223_assignment_partB.c -o partB
gauravghati@gauravghati:~/OS-Programming/assignment2-process$ ./partB

        ENTER SIZE OF ARRAY : 5

        ENTER ARRAY ELEMENTS : 1 9 2 6 4

        ENTER ELEMENT TO BE SEARCHED : 6

        MENU :
        1)HEAP SORT
        2)BUBBLE SORT
        ENTER YOUR CHOICE : 2

        FORK DONE

        ===========PARENT BLOCK============
        PARENT ID : 23321
        BUBBLE SORT
        SORTED ARRAY(PARENT) : 1 2 4 6 9

        FORK DONE

        ===========CHILD BLOCK============
        CHILD ID : 23324
        PARENT ID : 23321
        BUBBLE SORT
        SORTED ARRAY(CHILD) : 1 2 4 6 9
gauravghati@gauravghati:~/OS-Programming/assignment2-process$ No of arguments passed : 6
Arg[0] : 1
Arg[1] : 2
Arg[2] : 4
Arg[3] : 6
Arg[4] : 9
Arg[5] : 6
        ARGC : 6
Search : 6
        ELEMENT FOUND AT POSITION 3.
```

bsearch.c

```
/* Name-Gaurav Ghati
Div- TE10
Batch-L10
Rollno-33223*/
# include <stdio.h>
#include <stdlib.h>

int binarySearch(int arr[], int l, int r, int x)
{
if (r >= l) {
int mid = l + (r - l) / 2;
// If the element is present at the middle
// itself
if (arr[mid] == x)
return mid;
// If element is smaller than mid, then
// it can only be present in left subarray
if (arr[mid] > x)
return binarySearch(arr, l, mid - 1, x);
// Else the element can only be present
// in right subarray
return binarySearch(arr, mid + 1, r, x);
}
// We reach here when element is not
// present in array
return -1;
}
int main(int argc, char *argv[])
{
int i, arr[20],res, search, x;
printf("No of arguments passed : %d", argc);
for(int i=0; i<argc; i++) {
printf("\nArg[%d] : %s", i, argv[i]);
}
printf("\n\tARGC : %d", argc);
for(i=1; i<argc; i++) {

arr[i-1] = atoi(argv[i]);
}
search = arr[argc-2];
printf("\nSearch : %d", search);
//printf("\n\tENTER ELEMENT TO BE SEARCHED : ");
//scanf("%d", &search);
res = binarySearch(arr, 0, argc-3, search);
if(res==-1) printf("\n\tELEMENT NOT FOUND!\n");
else printf("\n\tELEMENT FOUND AT POSITION %d.\n", res+1);
}
```