Assignment 2

2a)

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

void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

void heapify(int arr[], int N, int i) {
  int largest = i;
  int l = 2 * i + 1;
  int r = 2 * i + 2;
  if (l < N && arr[l] > arr[largest]) largest = l;

  if (r < N && arr[r] > arr[largest]) largest = r;

  if (largest != i) {
    swap(&arr[i], &arr[largest]);

    heapify(arr, N, largest);
  }
}

void heapSort(int arr[], int N) {
  for (int i = N / 2 - 1; i >= 0; i--) heapify(arr, N, i);
  for (int i = N - 1; i > 0; i--) {
    swap(&arr[0], &arr[i]);
    heapify(arr, i, 0);
  }
}
```

```
void printArray(int arr[], int size) {
  for (int i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}

void Zombie() {
  pid_t pid = fork();

  if (pid < 0) {
    perror("fork failed");
    exit(1);
  } else if (pid == 0) {
    // Child process
    printf("Child process (PID: %d) running...\n", getpid());
    printf("Child process (PPID: %d) running...\n", getppid());
    exit(0);
  } else {
    // Parent process
    printf("Demonstrating zombie process.\n");
    printf("Parent process (PID: %d) running...\n", getpid());
    sleep(10);
    printf("Parent process (PID: %d) finished sleeping.\n", getpid());
  }
}

void Orphan() {
  pid_t pid = fork();
  if (pid < 0) {
    perror("fork failed");
    exit(1);
  } else if (pid == 0) {
    // Child process
    printf("Child process (PID: %d, PPID: %d) running...\n", getpid(),
```

```
            getppid());
    sleep(5);
    printf("Child process (PID: %d, PPID: %d) finished.\n", getpid(),
            getppid());
    exit(0);
  } else {
    printf("Demonstrating orphan process.\n");
    printf("Parent process (PID: %d) exiting...\n", getpid());
    exit(0);
  }
}

void SortByWaitCall(int arr[], int n) {
  pid_t pid = fork();

  if (pid < 0) {
    perror("fork failed");
    exit(1);
  } else if (pid == 0) {
    // Child process
    printf("Child process sorting with Heap Sort...\n");
    heapSort(arr, n);
    printf("Child process sorted array: ");
    printArray(arr, n);
    printf("Child process (PID: %d) finished.\n", getpid());
    exit(0);
  } else {
    // Parent process
    printf("Parent process sorting with Heap Sort...\n");
    heapSort(arr, n);
    printf("Array sorted and wait called\n ");

    // Wait for the child process to finish
    int status;
    pid_t child_pid = wait(&status);
```

```
    if (child_pid < 0) {
      perror("wait failed");
    } else {
      printf("Parent process (PID: %d) waited for child process (PID:
%d)\n",
             getpid(), child_pid);
    }
  }
}

int main() {
  int n, i, c;
  printf("Enter number of integers to sort: ");
  scanf("%d", &n);

  int arr[n];
  printf("Enter the integers:\n");
  for (i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
  }

  printf("Enter choice: 1. Zombie 2. Orphan 3. Using wait and sort\n");
  scanf("%d", &c);

  switch (c) {
    case 1:
      Zombie();
      break;

    case 2:
      Orphan();
      break;

    case 3:
      SortByWaitCall(arr, n);
      break;
```

```
    default:
      printf("Invalid choice\n");
      break;
  }

  return 0;
}
```

Output

```
Enter number of integers to sort: 5
Enter the integers:
48 79 12 23 40
Enter choice: 1. Zombie 2. Orphan 3. Using wait and sort
1
Demonstrating zombie process.
Parent process (PID: 1016) running...
Child process (PID: 1074) running...
Child process (PPID: 1016) running...
Parent process (PID: 1016) finished sleeping.

Enter number of integers to sort: 5
Enter the integers:
1 2 5 4 5
Enter choice: 1. Zombie 2. Orphan 3. Using wait and sort
2
Demonstrating orphan process.
Parent process (PID: 1162) exiting...
Child process (PID: 1239, PPID: 1162) running...
Child process (PID: 1239, PPID: 354) finished.

Enter number of integers to sort: 5
Enter the integers:
4 5 6 7 5
Enter choice: 1. Zombie 2. Orphan 3. Using wait and sort
3
```

```
Parent process sorting with Heap Sort...
Array sorted and wait called
Child process sorting with Heap Sort...
Child process sorted array: 4 5 5 6 7
Child process (PID: 1340) finished.
Parent process (PID: 1308) waited for child process (PID: 1340)
```

2b) main.c

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


void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

void heapify(int arr[], int N, int i) {
  int largest = i;
  int l = 2 * i + 1;
  int r = 2 * i + 2;
  if (l < N && arr[l] > arr[largest]) largest = l;
  if (r < N && arr[r] > arr[largest]) largest = r;

  if (largest != i) {
    swap(&arr[i], &arr[largest]);

    heapify(arr, N, largest);
  }
}
```

```
void heapSort(int arr[], int N) {
  for (int i = N / 2 - 1; i >= 0; i--) heapify(arr, N, i);

  for (int i = N - 1; i > 0; i--) {
    swap(&arr[0], &arr[i]);
    heapify(arr, i, 0);
  }
}

int main(){
    printf("Fork initialising\n");
    int n;

    printf("Enter number of integers to sort: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the integers:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    heapSort(arr, n);
    printf("Array sorted: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    pid_t pid = fork();

    if (pid < 0) {
        printf("Error occurred\n");
    } else if (pid == 0) {
        printf("Inside child process\n");
```

```
        char *argv[n + 2];
        argv[0] = (char *)"./hello";

        // Convert integers to strings
        for (int i = 0; i < n; i++) {
            argv[i + 1] = (char *)malloc(20 * sizeof(char)); // allocate
memory for each argument
            snprintf(argv[i + 1], 20, "%d", arr[i]);
        }
        argv[n + 1] = NULL;
        execv(argv[0], argv);
    } else {
        sleep(10);
        printf("Inside parent process\n");
    }
    return 0;
}
```

Hello.c

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

int main(int argc, char *argv[]) {
    printf("exec executed\n");
    printf("The PID of this process is: %d\n", getpid());

    if (argc > 1) {
        printf("Arguments received in reverse order: ");
        for (int i = argc - 1; i > 0; --i) {
            printf("%s ", argv[i]);
        }
        printf("\n");
    } else {
```

```
        printf("No arguments received.\n");
    }

    return 0;
}
```

Output

```
swikar@LAPTOP-3VLQDHIH:~$ g++ hello.c -o hello
swikar@LAPTOP-3VLQDHIH:~$ ./hello
exec executed
The PID of this process is: 2207
No arguments received.

swikar@LAPTOP-3VLQDHIH:~$ g++ a2b.c
swikar@LAPTOP-3VLQDHIH:~$ ./a.out
Fork initialising
Enter number of integers to sort: 5
Enter the integers:
2 1 4 5 7
Array sorted: 1 2 4 5 7
Inside child process
exec executed
The PID of this process is: 2336
Arguments received in reverse order: 7 5 4 2 1
Inside parent process
```