Assignment-4

Question-1

Concurrent merge sort using threads and processes and their relative comparison. I am doing selection sort when the number of elements to be sorted less than 5.

Normal merge sort

```
void  normal_mergesort(int  arr[],  int  l,  int  r)
{
    int  m;
    if (r  -  l  >=  4)
    {
        m  = (l  +  r) /  2;
        normal_mergesort(arr, l, m);
        normal_mergesort(arr, m  +  1, r);
        merge(arr, l, m, r);
    }
    else
    {
        selectionsort(arr, l, r);
    }
}
```

Explanation

This is just a regular merge sort code in which I am recursively calling the same function if the number of elements to be sorted greater than or equal to 5; otherwise, I will do a selection sort.

Concurrent merge sort using processes

```
void  mergesort(int  *arr,  int  low,  int  high)
{
    if (high  -  low  >=  4)
    {
        int  mid  = (low  +  high) /  2;
        int  pid1  =  fork();
        int  pid2;
        if (pid1  ==  0)
        {
            mergesort(arr, low, mid);
            _exit(0);
        }
        else
        {
```

```
            pid2  =  fork();
            if (pid2  ==  0)
            {
                mergesort(arr, mid  +  1, high);
                _exit(0);
            }
            else
            {
                int  status;
                waitpid(pid1, &status, 0);
                waitpid(pid2, &status, 0);
                merge(arr, low, mid, high);
            }
        }
    }
    else
    {
        selectionsort(arr, low, high);
    }
}
```

Explanation

This is the code for concurrent merge sort using processes where I am forking child process for each half of the array to be sorted, and the parent process will merge the two sorted halves returned by child processes.

Concurrent merge sort using threads

```
void  *threaded_mergesort(void  *a)
{
    struct  arg  *args  = (struct  arg  *)a;

    int  l  =  args->l;
    int  r  =  args->r;
    int  *arr  =  args->arr;
    if (l  >=  r)
    {
        return  NULL;
    }
    else  if (r  -  l  <  4)
    {
        selectionsort(arr, l, r);
        return  NULL;
    }
    int  m  = (l  +  r) /  2;
    struct  arg  a1;
    a1.l  =  l;
    a1.r  =  m;
    a1.arr  =  arr;
    pthread_t  tid1;
```

```
    pthread_create(&tid1, NULL, threaded_mergesort, &a1);
    struct  arg  a2;
    a2.l  =  m  +  1;
    a2.r  =  r;
    a2.arr  =  arr;
    pthread_t  tid2;
    pthread_create(&tid2, NULL, threaded_mergesort, &a2);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    merge(arr, l, m, r);
}
```

Explanation

This is the code for concurrent merge sort using threads where I am spawning child thread for each half of the array to be sorted, and the parent thread will merge the two sorted halves returned by child threads.

Analysis

N = 10

```
Normal mergesort ran:
        [ 144.422917 ] times faster than concurrent mergesort
        [ 118.475100 ] times faster than threaded concurrent mergesort
```

N = 100

```
Normal mergesort ran:
        [ 178.412992 ] times faster than concurrent mergesort
        [ 92.603488 ] times faster than threaded concurrent mergesort
```

N = 10000

```
Normal mergesort ran:
        [ 13.214972 ] times faster than concurrent mergesort
        [ 7.061855 ] times faster than threaded concurrent mergesort
```

Conclusion

If the number of elements to be sorted is less, then the overhead of OS to create the child processes or child threads due to which normal merge sort runs faster as compared to concurrent merge sort will become significant, but when the size of the input array increases then the overhead will start becoming

less significant as compared to the time required to sort the elements, so concurrent merge sort using threads or processes start running faster because these two also exploit parallelism. Concurrent merge sort using threads is faster than concurrent merge sort using processes because the creating of threads requires less time than the creation of processes because they share global data (text, data, and heap segments). In contrast, the child process has an isolated virtual address space.

Thank you!