

concurrent_merge sort (by forking):

if the size of current part to be sorted is less than 5 then i sorted it using selection sort other wise i created a child for left half, other child to sort the right half and their parent waits for both of them to get sorted. After left and right half got sorted they will exit and their parent merge the two sorted half.

```
if (r - l + 1 <= 4)
{
    selectionsort(arr, l, r);
    return;
}
```

```
else
{
    int status;
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);
    merge(arr, l, m, r);
}
```

concurrent_merge sort (by threads):

if the size of part of array to be sorted by a thread is less than 5 that thread will sort using select sorta and returns otherwise it creates two threads one for left half, one for right half and the main thread thread waits for two threads to get completed and then merges the result.

```
179 struct arg a1;
180 a1.l = l;
181 a1.r = m;
182 a1.arr = arr;
183 pthread_t tid1;
184 pthread_create(&tid1, NULL, threaded_mergesort, &a1);
185 struct arg a2;
186 a2.l = m + 1;
187 a2.r = r;
188 a2.arr = arr;
189 pthread_t tid2;
190 pthread_create(&tid2, NULL, threaded_mergesort, &a2);
191 //wait for two halves get sorted
192 pthread_join(tid1, NULL);
193 pthread_join(tid2, NULL);
194 merge(arr, l, m, r);
```

normal merge sort ran faster than concurrent process by threads or by child process.

```
hp@hp-HP-Pavilion-Laptop-15-cs2xxx:~/as4$ ./a.out
5
5 4 3 2 1
running concurrent mergesort for n = 5
1 2 3 4 5
time = 0.001116
Running threaded_mergesort for n = 5
1 2 3 4 5
time = 0.000754
Running normal merge_sort for n = 5
1 2 3 4 5
time = 0.000008
normal_mergesort ran:
[ 142.560663 ] times faster than concurrent_mergesort
[ 96.349887 ] times faster than threaded_concurrent_mergesort
```