Sorting of array

We need to sort an integer array recursively using mergesort. If the number of elements in an array at a point is less than 5, we need to perform a selection sort. The code's objective is to sort a given array by three different methods listed below and compare the time required for the completion for different values of n and varying input arrays.

Henceforth, the number of elements in the array is referred to as n.

Normal Mergesort

An array is sorted by recursively splitting up a parent array into two child arrays and merging them back to get a sorted result.

- **Recursive splitting of arrays**

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

  This function recursively splits the parent array into right and left half and merges them back. If n < 5, the selection sort function is called.

- **Merging the left anf right halves.**

```
void merge(int arr[], int l, int m, int r)
{
  int i, j, k;
  int n1 = m - l + 1;
  int n2 = r - m;
  int L[n1], R[n2];
  for (i = 0; i < n1; i++){
      L[i] = arr[l + i];
  }
  for (j = 0; j < n2; j++){
      R[j] = arr[m + 1 + j];
  }
  i = 0;
  j = 0;
  k = l;
  while (i < n1 && j < n2) {
      if (L[i] <= R[j]) {
          arr[k] = L[i];
          i++;
```

```
      }
      else {
          arr[k] = R[j];
          j++;
      }
      k++;
  }
  while (i < n1) {
      arr[k] = L[i];
      i++;
      k++;
  }
  while (j < n2) {
      arr[k] = R[j];
      j++;
      k++;
  }
}
```

This function merges the sort right and left subarrays into a sorted array.

- **Selection sort for number of integers < 5**

```
void swap(int *xp, int *yp)
{
  int temp = *xp;
  *xp = *yp;
  *yp = temp;
}
void selectionsort(int arr[], int l, int r)
{
  int i, j, min_idx;
  for (i = l; i < r-1; i++){
      min_idx = i;
      for (j = i+1; j < r; j++)
      if (arr[j] < arr[min_idx])
          min_idx = j;
      swap(&arr[min_idx], &arr[i]);
  }
}
```

When the number of integers in the array is less than 5, the function selectionsort performs the selection sort operation on the array. swap function swaps the integers at indices denoted by xp and yp;

Concurrent Mergesort

Mergesort is performed by recursively creating child processes for each half of the array and then merging them back to give a sorted parent array. Each child process has a different memory block associated with it, so they do not share variables unless explicitly asked. We need to create a shared memory block so that changes made to the array by each process can be ---------- by other processes as well.

- **Shared memory**

```
key_t mem_key = IPC_PRIVATE;
int shm_id = shmget(mem_key, sizeof(int)*(n+1), IPC_CREAT | 0666);
int *arr = shmat(shm_id, NULL, 0);
```

shmget allocates a System V shared memory segment, and shmat attaches the System V shared memory segment identified by shmid to the address space of the calling process.

- **Splitting the array and creating child process for each half**

```
void concurrent_mergesort(int arr[], int l, int r){
  if (r-l > 4) {
      int m = l + (r - l) / 2;
      int x = fork();
      if(x==0){
          concurrent_mergesort(arr, l, m);
          exit(0);
      }
      else{
              int y = fork();
               if(y==0){
                   concurrent_mergesort(arr, m + 1, r);
                   exit(0);
               }
               else{
                  int status;
                  waitpid(x, &status, 0);
                  waitpid(y, &status, 0);
                   merge(arr, l, m, r);
               }
      }
  }
  else{
      int x = fork();
      if(x==0){
          selectionsort(arr, l, r+1);
          exit(0);
      }
      else{
              int status;
              waitpid(x, &status, 0);
      }
  }
}
```

This function checks the value of n and creates a child process(es) depending on whether n < 5 or n>=5. For n >= 5, two children processes are created, one for the left half and the other for the right half. Once they are sorted, they are merged into a single array. For n < 5, selection sort is performed.

Threaded Mergesort

A thread is a single sequence stream within a process. Threads are not independent of one other like processes; thus, threads share with other threads their code section, data section, and OS resources like open files and signals. Like the process, a thread has its own program counter (PC), a register set, and a stack space. Mergesort can be performed by recursively creating a thread for each subarray.

- **Threaded Mergesort**

```
void *threaded_mergesort(void *a){
  struct arg *args = (struct arg*) a;
  int l = args->l;
  int r = args->r;
  int *arr = args->array;

  if (r-l > 4) {
      int m = l + (r - l) / 2;
      struct arg a_prime1;
      a_prime1.l = l;
      a_prime1.r = m;
      a_prime1.array = arr;
      pthread_t tid1;
      pthread_create(&tid1, NULL, threaded_mergesort, &a_prime1);

      struct arg a_prime2;
      a_prime2.l = m+1;
      a_prime2.r = r;
      a_prime2.array = arr;
      pthread_t tid2;
      pthread_create(&tid2, NULL, threaded_mergesort, &a_prime2);

      pthread_join(tid1, NULL);
      pthread_join(tid2, NULL);

      merge(arr, l, m, r);
  }
  else{
      struct arg a_prime1;
      a_prime1.l = l;
      a_prime1.r = r + 1;
      a_prime1.array = arr;
      pthread_t tid1;
      pthread_create(&tid1, NULL, threaded_selectionsort, &a_prime1);
      pthread_join(tid1, NULL);
  }
}
```

  This function creates thread(s), depending on the value of n. For n < 5, a single thread is created that performs selection sort, the array is recursively split, and a separate thread is created for each subarray. Then they are merged back.

- **Threaded selection sort for n<5**

```
void *threaded_selectionsort(void *a){
  struct arg *args = (struct arg*) a;
  int l = args->l;
  int r = args->r;
  int *arr = args->array;
  int i, j, min_idx;
  for (i = l; i < r-1; i++){
      min_idx = i;
      for (j = i+1; j < r; j++)
      if (arr[j] < arr[min_idx])
          min_idx = j;
      swap(&arr[min_idx], &arr[i]);
  }
}
```

This function takes input as an array and performs selection sort on it for threads.

Comparison of the performance

Normal mergesort is always faster than the other two.