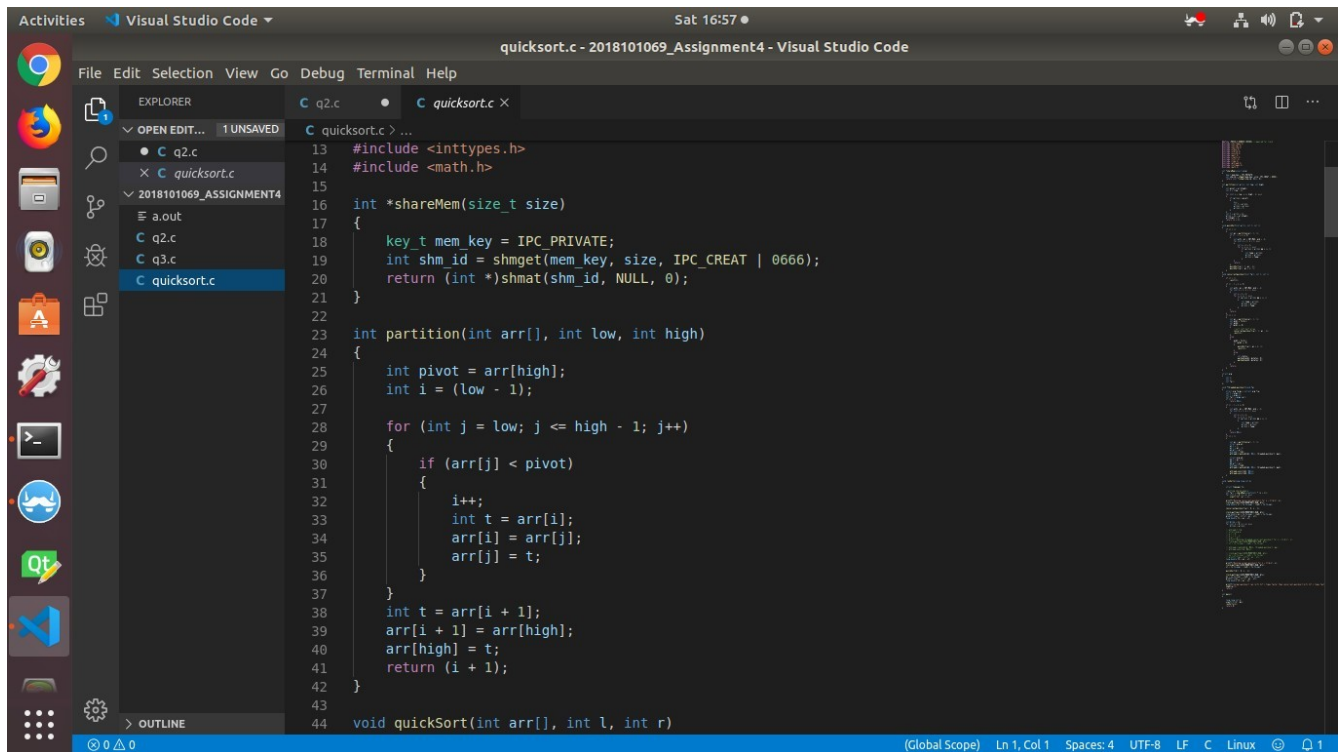


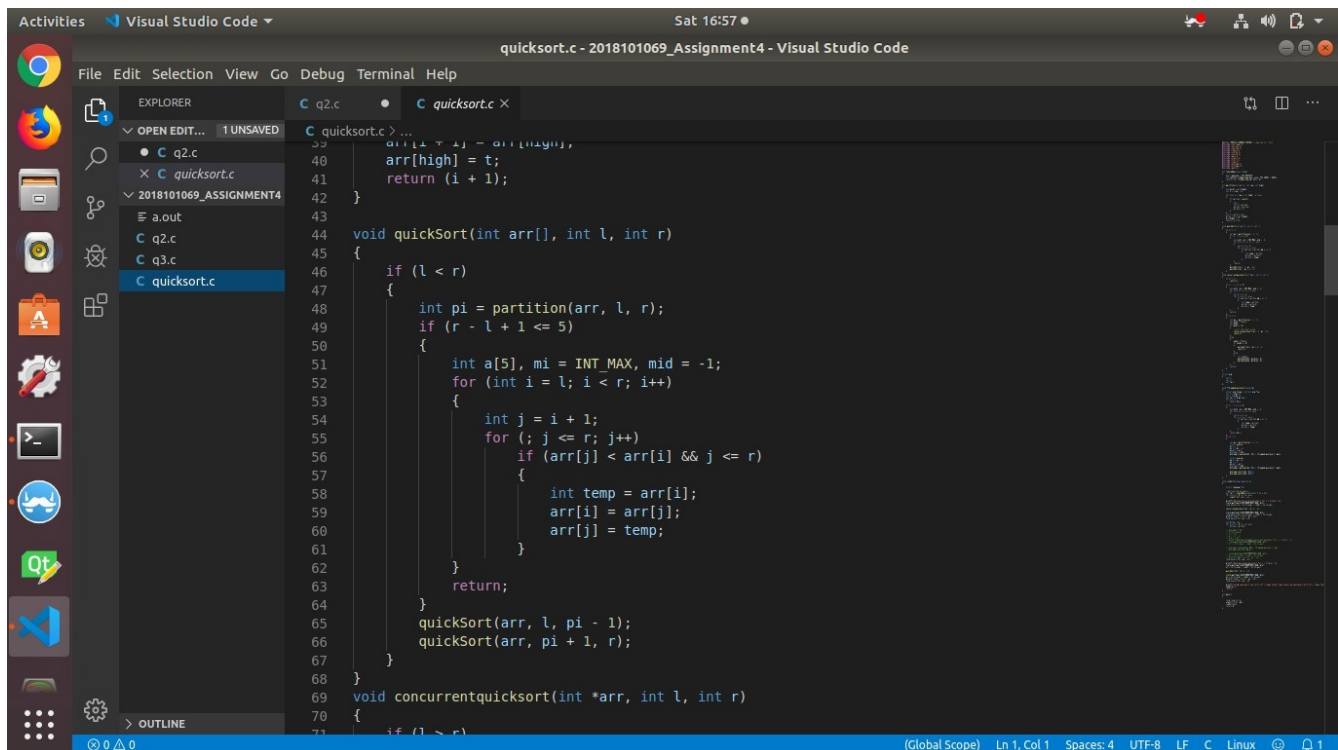
Report for question 1



The screenshot shows the Visual Studio Code editor with the file `quicksort.c` open. The code defines a `shareMem` function to create shared memory and a `partition` function to divide an array. The `partition` function uses a pivot at the high index and moves elements less than the pivot to the left.

```
13 #include <inttypes.h>
14 #include <math.h>
15
16 int *shareMem(size_t size)
17 {
18     key_t mem_key = IPC_PRIVATE;
19     int shm_id = shmget(mem_key, size, IPC_CREAT | 0666);
20     return (int *)shmat(shm_id, NULL, 0);
21 }
22
23 int partition(int arr[], int low, int high)
24 {
25     int pivot = arr[high];
26     int i = (low - 1);
27
28     for (int j = low; j <= high - 1; j++)
29     {
30         if (arr[j] < pivot)
31         {
32             i++;
33             int t = arr[i];
34             arr[i] = arr[j];
35             arr[j] = t;
36         }
37     }
38     int t = arr[i + 1];
39     arr[i + 1] = arr[high];
40     arr[high] = t;
41     return (i + 1);
42 }
43
44 void quickSort(int arr[], int l, int r)
```

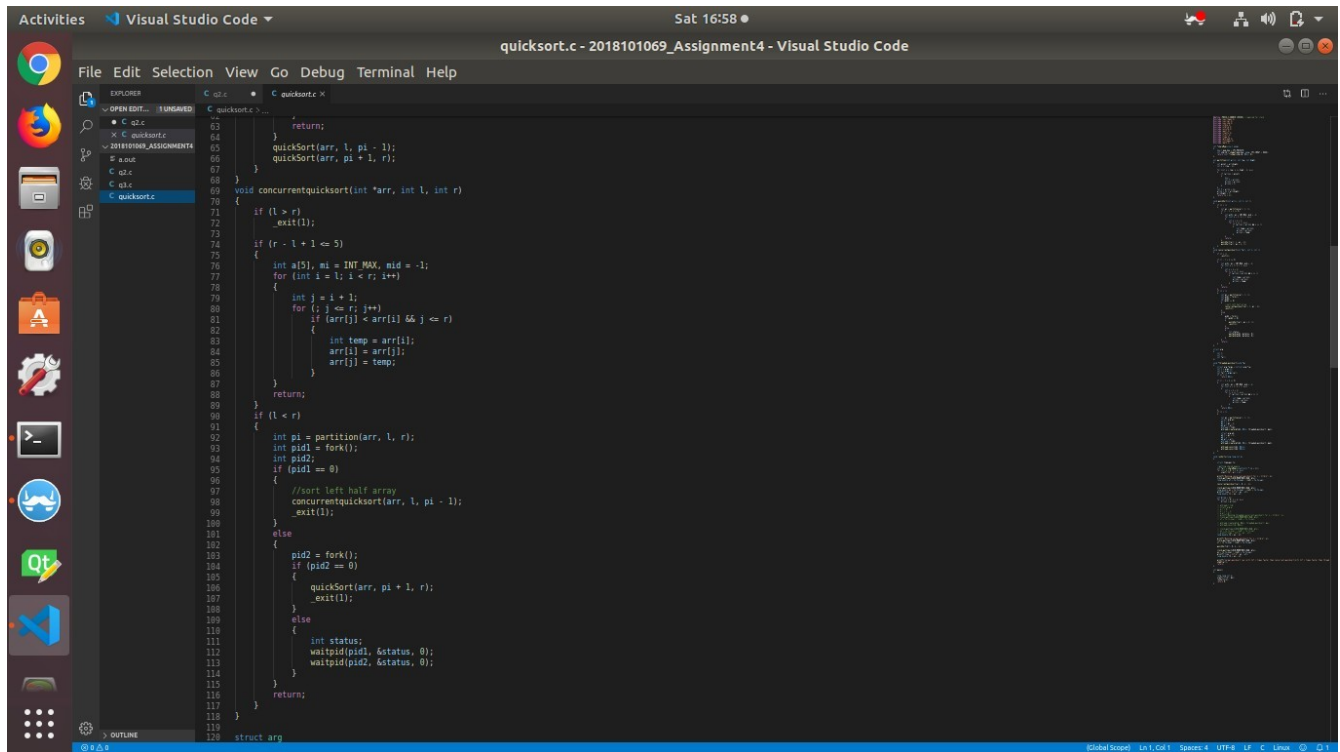
ShareMem function is used to create a shared memory array used in concurrent and threaded quicksort. Partition function is used to get array divide into 2 parts.



The screenshot shows the continuation of the `quicksort.c` file. It includes the `quickSort` function, which recursively calls `partition` and `quickSort` on the left and right sub-arrays. It also shows the beginning of the `concurrentquickSort` function.

```
39 arr[i + 1] = arr[high];
40 arr[high] = t;
41 return (i + 1);
42 }
43
44 void quickSort(int arr[], int l, int r)
45 {
46     if (l < r)
47     {
48         int pi = partition(arr, l, r);
49         if (r - l + 1 <= 5)
50         {
51             int a[5], mi = INT_MAX, mid = -1;
52             for (int i = l; i < r; i++)
53             {
54                 int j = i + 1;
55                 for (; j <= r; j++)
56                 {
57                     if (arr[j] < arr[i] && j <= r)
58                     {
59                         int temp = arr[i];
60                         arr[i] = arr[j];
61                         arr[j] = temp;
62                     }
63                 }
64                 return;
65             }
66             quickSort(arr, l, pi - 1);
67             quickSort(arr, pi + 1, r);
68         }
69     }
70
71 void concurrentquickSort(int *arr, int l, int r)
72 {
73     if (l > r)
```

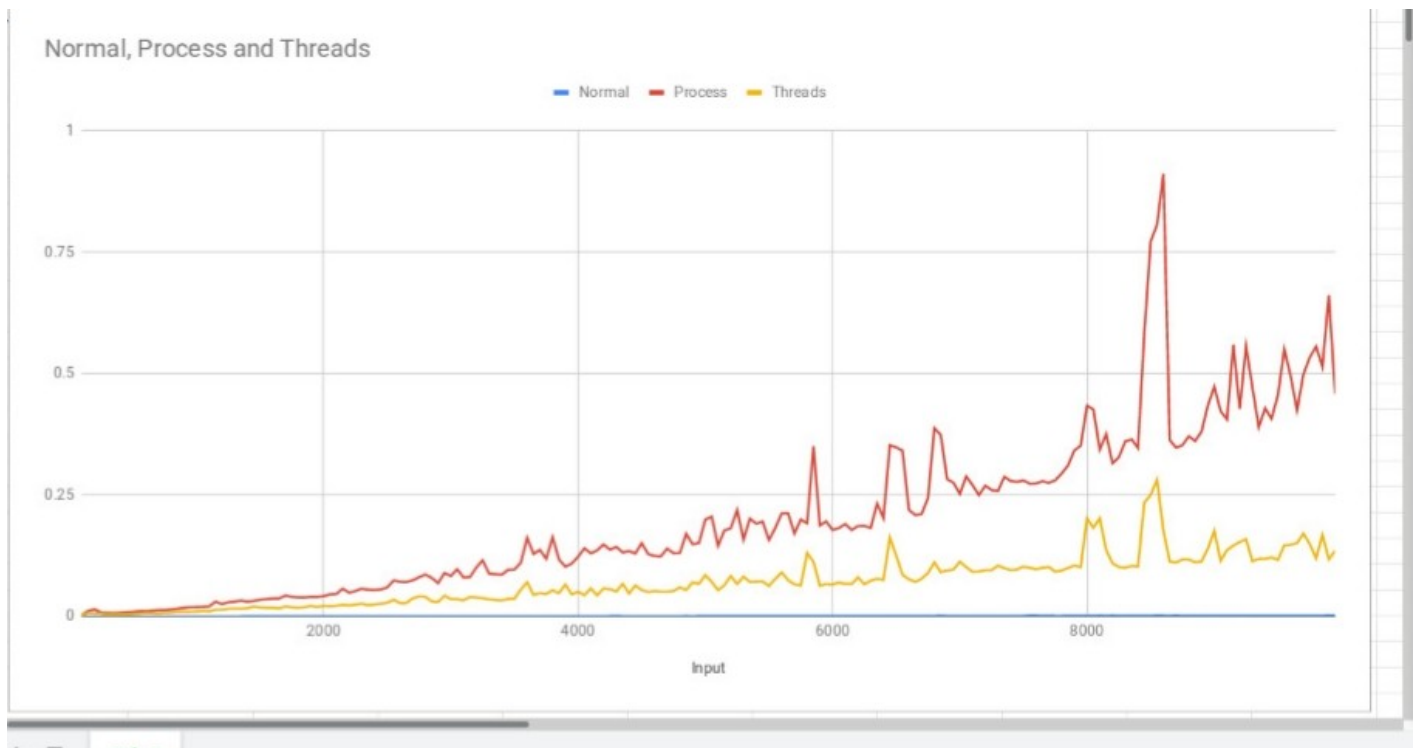
This snippet shows quicksort function which is used to sort using normal quick sort algorithm.



The screenshot shows the Visual Studio Code editor with a C program named 'quicksort.c'. The code implements a normal quicksort algorithm. It includes a partition function that selects a pivot and rearranges the array elements. The main function 'quicksort' uses a recursive approach to sort the array. The code is as follows:

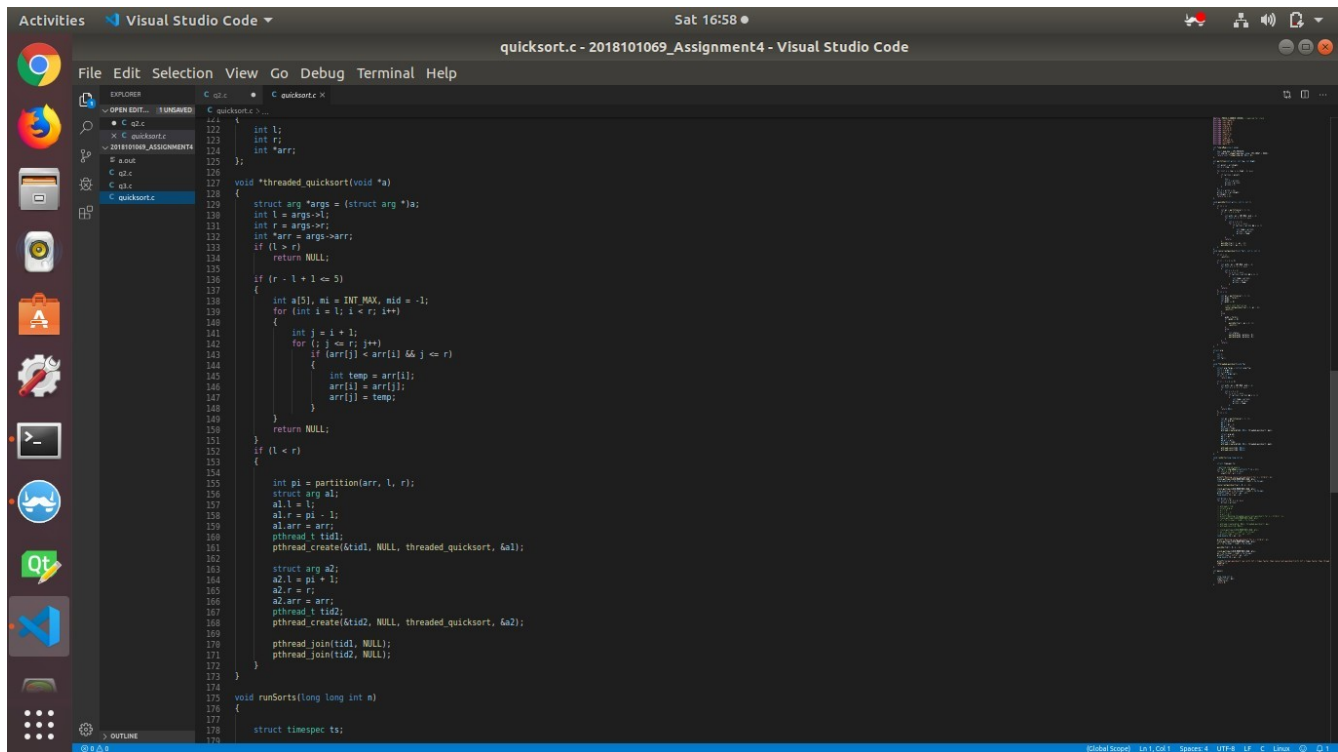
```
1 // Normal Quick Sort
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int partition(int arr[], int l, int r) {
6     int pivot = arr[r];
7     int i = l - 1;
8     for (int j = l; j < r; j++) {
9         if (arr[j] < pivot) {
10             i++;
11             swap(arr[i], arr[j]);
12         }
13     }
14     swap(arr[i + 1], arr[r]);
15     return i + 1;
16 }
17
18 void quicksort(int arr[], int l, int r) {
19     if (l < r) {
20         int pi = partition(arr, l, r);
21         quicksort(arr, l, pi - 1);
22         quicksort(arr, pi + 1, r);
23     }
24 }
25
26 int main() {
27     int arr[] = {10, 8, 6, 4, 2, 1, 3, 5, 7, 9};
28     int n = sizeof(arr) / sizeof(arr[0]);
29     quicksort(arr, 0, n - 1);
30     for (int i = 0; i < n; i++) {
31         printf("%d ", arr[i]);
32     }
33     printf("\n");
34     return 0;
35 }
```

This snippet shows concurrent quicksort function which is used to create parallel process and run quick sort.



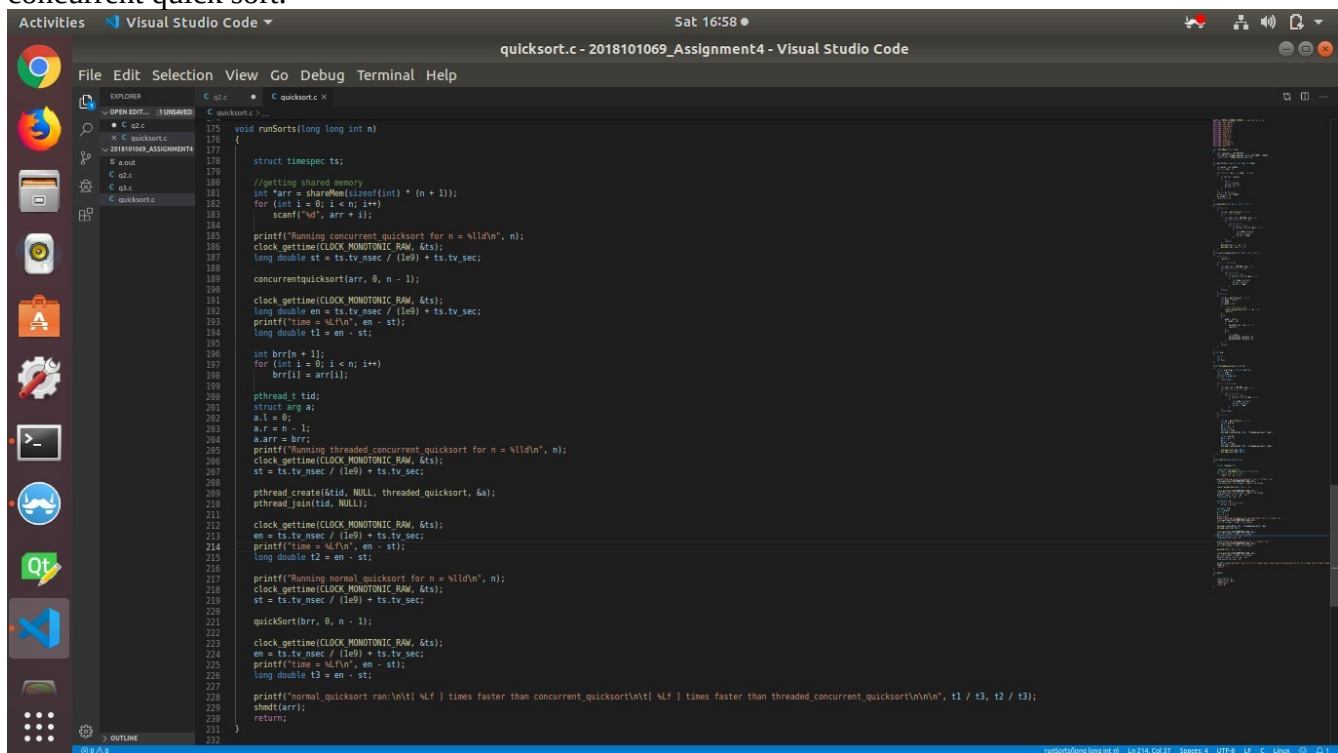
This snippet is used for quicksort algorithm using threading to get final quick sort algorithm

It will create thread to compute sorted array for divided part by giving each thread a subproblem.



```
122 int l;  
123 int r;  
124 int *arr;  
125 };  
126  
127 void *threaded_quicksort(void *a)  
128 {  
129     struct arg *args = (struct arg *)a;  
130     int l = args->l;  
131     int r = args->r;  
132     int *arr = args->arr;  
133     if (l > r)  
134         return NULL;  
135  
136     if (r - l + 1 <= 5)  
137     {  
138         int a[5], m1 = INT_MAX, mid = -1;  
139         for (int i = l; i <= r; i++)  
140         {  
141             int j = i + 1;  
142             for (; j <= r; j++)  
143             {  
144                 if (arr[i] < arr[j])  
145                 {  
146                     int temp = arr[i];  
147                     arr[i] = arr[j];  
148                     arr[j] = temp;  
149                 }  
150             }  
151             return NULL;  
152         }  
153     }  
154  
155     int pi = partition(arr, l, r);  
156     struct arg a1;  
157     a1.l = l;  
158     a1.r = pi - 1;  
159     a1.arr = arr;  
160     pthread_t tid1;  
161     pthread_create(&tid1, NULL, threaded_quicksort, &a1);  
162  
163     struct arg a2;  
164     a2.l = pi + 1;  
165     a2.r = r;  
166     a2.arr = arr;  
167     pthread_t tid2;  
168     pthread_create(&tid2, NULL, threaded_quicksort, &a2);  
169  
170     pthread_join(tid1, NULL);  
171     pthread_join(tid2, NULL);  
172 }  
173  
174 void runSorts(long long int n)  
175 {  
176     struct timespec ts;  
177  
178     //getting shared memory  
179     int *arr = sharedMem(sizeof(int) * (n + 1));  
180     for (int i = 0; i < n; i++)  
181         scanf("%d", &arr[i]);  
182  
183     printf("Running concurrent quicksort for n = %lld\n", n);  
184     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
185     long double st = ts.tv_nsec / (1e9) + ts.tv_sec;  
186  
187     concurrentquicksort(arr, 0, n - 1);  
188  
189     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
190     long double en = ts.tv_nsec / (1e9) + ts.tv_sec;  
191     printf("time = %Lf\n", en - st);  
192     long double t1 = en - st;  
193  
194     int brr[n + 1];  
195     for (int i = 0; i < n; i++)  
196         brr[i] = arr[i];  
197  
198     pthread_t tid;  
199     struct arg a;  
200     a.l = 0;  
201     a.r = n - 1;  
202     a.arr = brr;  
203     printf("Running threaded concurrent quicksort for n = %lld\n", n);  
204     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
205     st = ts.tv_nsec / (1e9) + ts.tv_sec;  
206  
207     pthread_create(&tid, NULL, threaded_quicksort, &a);  
208     pthread_join(tid, NULL);  
209  
210     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
211     en = ts.tv_nsec / (1e9) + ts.tv_sec;  
212     printf("time = %Lf\n", en - st);  
213     long double t2 = en - st;  
214  
215     printf("Running normal quicksort for n = %lld\n", n);  
216     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
217     st = ts.tv_nsec / (1e9) + ts.tv_sec;  
218  
219     quickSort(brr, 0, n - 1);  
220  
221     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
222     en = ts.tv_nsec / (1e9) + ts.tv_sec;  
223     printf("time = %Lf\n", en - st);  
224     long double t3 = en - st;  
225  
226     printf("normal quicksort ran %lf times faster than concurrent quicksort\n%lf times faster than threaded concurrent quicksort\n\n", t1 / t3, t2 / t3);  
227     shutdown(arr);  
228     return;  
229 }  
230  
231  
232 }
```

This code snippet shows time comparison between normal quicksort and threaded quicksort and concurrent quick sort.



```
175 void runSorts(long long int n)  
176 {  
177     struct timespec ts;  
178  
179     //getting shared memory  
180     int *arr = sharedMem(sizeof(int) * (n + 1));  
181     for (int i = 0; i < n; i++)  
182         scanf("%d", &arr[i]);  
183  
184     printf("Running concurrent quicksort for n = %lld\n", n);  
185     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
186     long double st = ts.tv_nsec / (1e9) + ts.tv_sec;  
187  
188     concurrentquicksort(arr, 0, n - 1);  
189  
190     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
191     long double en = ts.tv_nsec / (1e9) + ts.tv_sec;  
192     printf("time = %Lf\n", en - st);  
193     long double t1 = en - st;  
194  
195     int brr[n + 1];  
196     for (int i = 0; i < n; i++)  
197         brr[i] = arr[i];  
198  
199     pthread_t tid;  
200     struct arg a;  
201     a.l = 0;  
202     a.r = n - 1;  
203     a.arr = brr;  
204     printf("Running threaded concurrent quicksort for n = %lld\n", n);  
205     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
206     st = ts.tv_nsec / (1e9) + ts.tv_sec;  
207  
208     pthread_create(&tid, NULL, threaded_quicksort, &a);  
209     pthread_join(tid, NULL);  
210  
211     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
212     en = ts.tv_nsec / (1e9) + ts.tv_sec;  
213     printf("time = %Lf\n", en - st);  
214     long double t2 = en - st;  
215  
216     printf("Running normal quicksort for n = %lld\n", n);  
217     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
218     st = ts.tv_nsec / (1e9) + ts.tv_sec;  
219  
220     quickSort(brr, 0, n - 1);  
221  
222     clock_gettime(CLOCK_MONOTONIC_RAW, &ts);  
223     en = ts.tv_nsec / (1e9) + ts.tv_sec;  
224     printf("time = %Lf\n", en - st);  
225     long double t3 = en - st;  
226  
227     printf("normal quicksort ran %lf times faster than concurrent quicksort\n%lf times faster than threaded concurrent quicksort\n\n", t1 / t3, t2 / t3);  
228     shutdown(arr);  
229     return;  
230 }  
231  
232 }
```