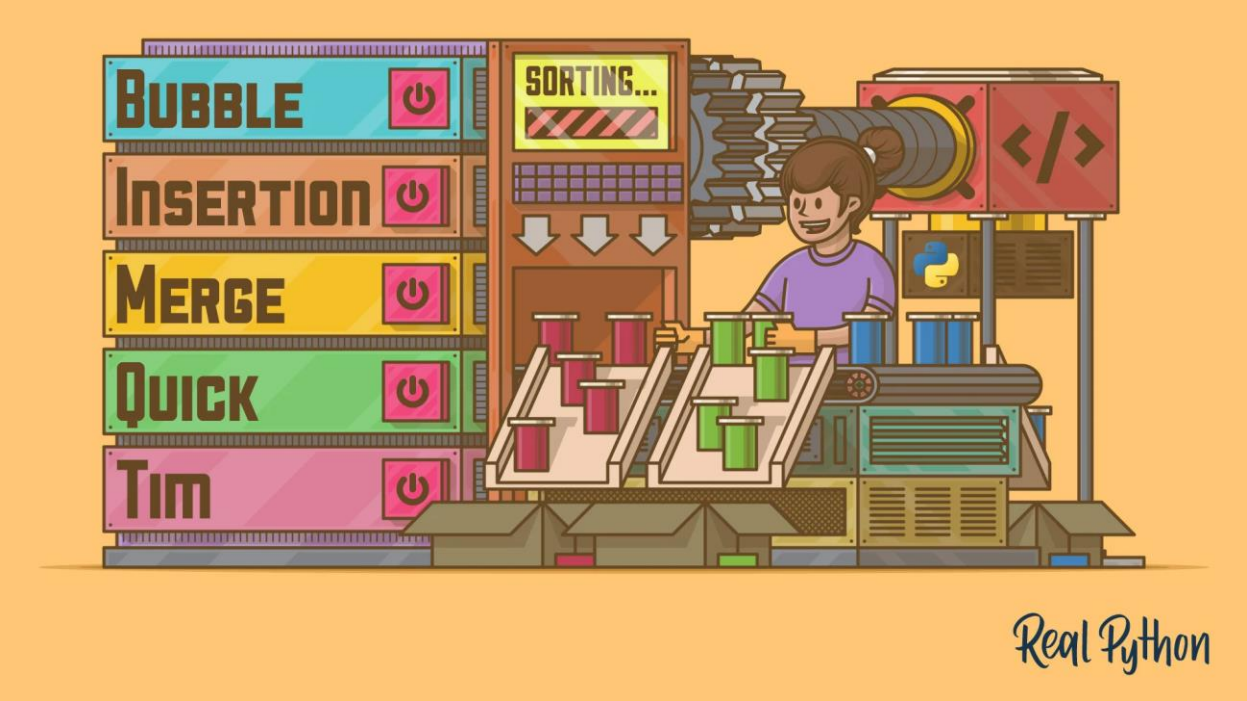
OS PROJECT REPORT



Group Members:

Section 5C

Bilal Shafiq (21K-3222)

Muhammad Haris (21K-3415)

Ghufran Ghous (21I-2991)

Department of Computer Science

National University of Computer and Emerging Sciences-FAST

Karachi Campus

Parallel Programming Comparison of Sorting Algorithms using Pthreads vs OpenMP vs Serial [3 Algorithms]

INTRODUCTION

This project is part of our Operating Systems Course, we chose to work on the topic of 'Parallel Programming Comparison of Sorting Algorithms Using Pthreads vs OpenMP vs. Serial' using heap sort, radix sort and count sort.

We implemented a system that allowed us to differentiate the time required to execute different sorting algorithms using several approaches such as Pthreads, OpenMP for Multithreading, and Serial processing. The execution time was then analyzed of each algorithm using bar graphs which we plotted using pyplot on Google Colab to determine which is the most efficient.

MOTIVATION

In the realm of operating systems, the optimization of computational processes is paramount. The motivation behind this project lies in the exploration and comparison of sorting algorithms, namely Heap Sort, Radix Sort, and Count Sort, under various execution models. The focus is on understanding the impact of parallel programming using Pthreads and OpenMP in comparison to the traditional serial processing. As the demand for efficient and scalable algorithms continues to grow, this project aims to contribute insights into the performance of sorting algorithms in parallel environments.

PROBLEM STATEMENT

The challenge addressed in this project is to evaluate and compare the efficiency of sorting algorithms—Heap Sort, Radix Sort, and Count Sort—under different execution models. The project seeks to understand the implications of parallel programming using Pthreads and OpenMP in contrast to the traditional serial processing for these sorting algorithms. The problem is to ascertain the impact of concurrency and parallelism on the execution time of sorting algorithms, ultimately providing a comprehensive comparison of their performance.

CONTRIBUTION

This project contributes to the field of operating systems by providing a detailed analysis of sorting algorithms under distinct execution models. The contributions include:

1. **Implementation of Sorting Algorithms:**
    - Development and implementation of Heap Sort, Radix Sort, and Count Sort in the C language.
2. **Parallel Programming Models:**
    - Utilization of Pthreads and OpenMP for parallelizing the sorting algorithms, enabling a comparative study of their performance.
3. **Execution Time Analysis:**
    - Measurement and analysis of execution times for each sorting algorithm under serial, Pthreads, and OpenMP execution models.
4. **Visualization:**
    - Use of Python and Google Colab to create visual representations (bar graphs) of the execution times, facilitating a clear understanding of algorithmic efficiency.

METHODOLOGY

In this project, we have used the execution models; Serial, OpenMP, & PThreads where serial refers to executing one computation at a time and parallelism refers to the simultaneous execution of several calculations. Pthreads is an execution model that allows a program to control multiple different flows of work that overlap in time. OpenMP is an application programming interface for parallelizing sequential programs written in C on shared-memory platforms. It supports loop-level and function-level parallelism.

FEATURES

The main function calls a function to generate an array randomly dynamically. Then this array is sorted using sorting algorithms of Heap, Radix and Count Sort. Each is sorted 3 times using different methods of executions of Pthreads, Serial and OpenMP. For each method, time is computed and this is then compared using a graph in Python.
The tools we have utilized are as follows:
● C language
● Google Colab
● Linux (Ubuntu)

CODE SCREENSHOTS

[*] Heap.c

```
147 void heapify(int a[], int n, int i){
148     int max = i;
149     int left = 2*i;
150     int right = 2*i+1;
151         int temp;
152
153     if(left<n && a[left] > a[max]){
154         max = left;
155     }
156
157     if(right<n && a[right] > a[max]){
158         max = right;
159     }
160
161     if(max!=i){
162                     temp = a[i];
163                     a[i] = a[max];
164                     a[max] = temp;
165         //swap(a[i], a[max]);
166         heapify(a, n, max);
167     }
168 }
169
170 void *heapSort(void* arr){
171
172     int i, temp;
173        int* a= arr;
174
175
176     for(i=n/2-1; i>=0; i--){
177         heapify(a, n, i);
178     }
179
180     for(i=n-1; i>=0; i--){
181                     temp = a[0];
182                     a[0] = a[i];
183                     a[i] = temp;
184     //  swap(a[0], a[i]);
185         heapify(a, i, 0);
186     }
187
188        printf("\n\n\t\t\t\t\t\t\t\t\t\tSorted Array:
189     display(a, n);
190 }
```

[*] Heap.c

```
192 void heapS(int a[], int n){
193     int i, temp;
194
195
196     for(i=n/2-1; i>=0; i--){
197         heapify(a, n, i);
198     }
199
200     for(i=n-1; i>=0; i--){
201                     temp = a[0];
202                     a[0] = a[i];
203                     a[i] = temp;
204         heapify(a, i, 0);
205     }
206 }
207
208 void threads(){
209 FILE* fptr;
210 ms("\tP t h r e a d s");
211
212 double start, stop;
213
214    pthread_t t1;
215 start=clock();
216
217        pthread_create(&t1, NULL, heapSort, (void*)A);
218
219         pthread_join(t1, NULL);
220 stop=clock();
221
222 double time = (double)(stop-start)/CLOCKS_PER_SEC;
223 printf("\n\t\t\t\t\t\t\t\t\t\tTime Taken:       %lf seconds.\n",
224 (double)(stop-start)/CLOCKS_PER_SEC);
225
226 fptr = fopen("HeapTime.txt", "a");
227 if(fptr == NULL){
228 printf("\nError, file not created!");
229 exit(1);
230 }
231
232 fprintf(fptr, "\nPTHREADS:\n%f", time);
233 fclose(fptr);
234 }
```

[*] Heap.c

```
237 void openmp(){
238 FILE* fptr;
239 ms("\tO p e n M P");
240 double start, stop;
241 start = omp_get_wtime();
242
243 #pragma omp parallel sections num_threads(8)
244 {
245
246   #pragma omp section
247     {
248
249     heapS(B, m);
250
251      printf("\n\n\t\t\t\t\t\t\t\t\t\tSorted Array:      ");
252      display(B, m);
253     }
254 }
255 stop = omp_get_wtime();
256 double time;
257 time = stop-start;
258 printf("\n\t\t\t\t\t\t\t\t\t\tTime Taken:       %lf seconds.\n",
259 (stop-start));
260
261 fptr = fopen("HeapTime.txt", "a");
262 if(fptr == NULL){
263 printf("\nError, file not created!");
264 exit(1);
265 }
266
267 fprintf(fptr, "\nOPENMP:\n%f", time);
268 fclose(fptr);
269
270 }
271
272 int randArray(int *r){
273
274 srand(time(0));
275
276     for(int i=0; i<size; i++){
277        r[i]=rand()%100 + 1;
278     }
279 return *r;
280 }
```

[*] Heap.c

```
286 int main(){
287 messageA();
288 printf("\n\t\t\t\t\t\t\t\t\t\tPress Any Key to Continue");
289 getchar();
290 clrscr();
291     message();
292 FILE* fptr;
293
294  C = (int*) malloc(size*sizeof(int));
295   randArray(C);
296  printf("\t\t\t\t\t\t\t\t\t\t    Generating Array Randomly: \n\n");
297 displayA(C, size);
298
299     ms("S e r i a l  E x e c u t i o n");
300
301 double start, stop;
302 start=clock();
303 heapS(C, o);
304     printf("\n\n\t\t\t\t\t\t\t\t\t\tSorted Array:      ");
305 display(C, o);
306 stop=clock();
307 double time = (double)(stop-start)/CLOCKS_PER_SEC;
308
309 printf("\n\t\t\t\t\t\t\t\t\t\tTime Taken:       %lf seconds.\n",
310 (double)(stop-start)/CLOCKS_PER_SEC);
311
312    fptr = fopen("HeapTime.txt", "w");
313 if(fptr == NULL){
314 printf("\nError, file not created!");
315 exit(1);
316 }
317 fprintf(fptr, "Serial:\n%f", time);
318
319     fclose(fptr);
320
321  A = (int*) malloc(size*sizeof(int));
322   randArray(A);
323 threads();
324
325  B = (int*) malloc(size*sizeof(int));
326   randArray(B);
327       openmp();
328
329 }
```

[*] Radix.c Count.c

```c
int max_element(int array[], int n) {
  int max = array[0];
  for (int i = 1; i < n; i++)
    if (array[i] > max)
      max = array[i];
  return max;
}

void SF(int array[], int size, int sd){
  int output[size + 1];
  int max = (array[0] / sd) % 10;

  for (int i = 1; i < size; i++) {
    if (((array[i] / sd) % 10) > max)
      max = array[i];
  }
  int count[max + 1];

  for (int i = 0; i < max; ++i)
    count[i] = 0;

  for (int i = 0; i < size; i++)
    count[(array[i] / sd) % 10]++;


  for (int i = 1; i < 10; i++)
    count[i] += count[i - 1];

  for (int i = size - 1; i >= 0; i--) {
    output[count[(array[i] / sd) % 10] - 1] = array[i];
    count[(array[i] / sd) % 10]--;
  }

  for (int i = 0; i < size; i++)
    array[i] = output[i];
}

void *radixsort(void *a) {
    int *array=a;
  int max = max_element(array, n);
  for (int sd = 1; max / sd > 0; sd *= 10)
    SF(array, n, sd);
}
```

[*] Radix.c Count.c

```c
void radix_sort(int array[], int size) {
  int max = max_element(array, size);
  for (int sd = 1; max / sd > 0; sd *= 10)
    SF(array, size, sd);
}

void printArray(int array[], int size) {
size = 10;
  for (int i = 0; i < size; ++i) {
    printf("%d ", array[i]);
  }
  printf("\n");
}

void threads(){
FILE* fptr;
  ms("\tP t h r e a d s");

    double start, stop;
    pthread_t t2;

    start=clock();
    pthread_create(&t2, NULL, radixsort, (void*)A);

    pthread_join(t2, NULL);
    stop=clock();
 printf("\n\n\t\t\t\t\t\t\t\t\t\t\tSorted Array:     ");
    printArray(A, n);

printf("\t\t\t\t\t\t\t\t\t\tTime Taken:       %lf seconds.\n",
(double)(stop-start)/CLOCKS_PER_SEC);

double time = (double)(stop-start)/CLOCKS_PER_SEC;

fptr = fopen("RadixTime.txt", "a");
if(fptr == NULL){
printf("\nError, file not created!");
exit(1);
}

fprintf(fptr, "\nPTHREADS:\n%f", time);
fclose(fptr);
}
```

[*] Count.c

```c
int getMax(int a[], int n) {
    int max = a[0];
    for(int i = 1; i<n; i++) {
        if(a[i] > max)
            max = a[i];
    }
    return max;
  }
void countSort(int a[], int n)  {
    int output[n+1];
    int max = getMax(a, n);
    int count[max+1];

  for (int i = 0; i <= max; ++i)
  {
    count[i] = 0;
  }

  for (int i = 0; i < n; i++)
  {
    count[a[i]]++;
  }

   for(int i = 1; i<=max; i++)
     count[i] += count[i-1];

  for (int i = n - 1; i >= 0; i--) {
    output[count[a[i]] - 1] = a[i];
    count[a[i]]--;
}

    for(int i = 0; i<n; i++) {
      a[i] = output[i];
    }
}

void* countS(void *arr)  {
int *a=arr;

 int output[n+1];
   int max = getMax(a, n);
   int count[max+1];

  for (int i = 0; i <= max; ++i)
```

[*] Radix.c Count.c

```c
void openmp(){
FILE* fptr;
fflush(stdout);
    ms("\tO p e n M P");
fflush(stdout);
    double start, stop;
    start = omp_get_wtime();
    #pragma omp parallel sections num_threads(8)
    {
        #pragma omp section
        {
            radix_sort(B,m);
            printf("\n\n\t\t\t\t\t\t\t\t\t\t\tSorted Array:     ");
            printArray(B, m);

            }
      }
    stop = omp_get_wtime();
    printf("\t\t\t\t\t\t\t\t\t\tTime Taken:       %lf seconds.\n",
    (stop-start));
double time = stop-start;
fptr = fopen("RadixTime.txt", "a");
if(fptr == NULL){
printf("\nError, file not created!");
exit(1);
}

fprintf(fptr, "\nOpenMP:\n%f", time);
fclose(fptr);
}

int randArray(int *r){
    srand(time(0));

    for(int i=0; i<100; i++){
       r[i]=rand()%100 + 1;
    }
    return *r;
}

void displayA(int a[], int n){
     printf("\t\t\t");
    for(int i=0; i<50; i++){
          printf("%d ", a[i]);
```

[*] Count.c

```
  for (int i = 0; i <= max; ++i)   {
    count[i] = 0;
  }

  for (int i = 0; i < n; i++) {
    count[a[i]]++;
  }
   for(int i = 1; i<=max; i++)
     count[i] += count[i-1];

  for (int i = n - 1; i >= 0; i--) {
    output[count[a[i]] - 1] = a[i];
    count[a[i]]--;
}
   for(int i = 0; i<n; i++) {
     a[i] = output[i];
   }
}
void threads(){
FILE* fptr;

ms("\tP t h r e a d s");
double start, stop;
   pthread_t t1;
start=clock();
      pthread_create(&t1, NULL, countS, (void*)A);
       pthread_join(t1, NULL);
     printf("\n\n\t\t\t\t\t\t\t\t\t\t\tSorted Array:     ");
    display(A, n);
stop=clock();
printf("\n\t\t\t\t\t\t\t\t\t\tTime Taken:       %lf seconds.\n",
(double)(stop-start)/CLOCKS_PER_SEC);

double time = (double)(stop-start)/CLOCKS_PER_SEC;

fptr = fopen("CountTime.txt", "a");
if(fptr == NULL){
printf("\nError, file not created!");
exit(1);
}

fprintf(fptr, "\nPTHREADS:\n%f", time);
fclose(fptr);
}
```

OUTPUT SCREENSHOTS

```
k200406_fatima@Fatima-VirtualBox:~/Desktop/proj/final$ gcc -o h Heap.c -lpthread -fopenmp
k200406_fatima@Fatima-VirtualBox:~/Desktop/proj/final$ ./h

                    S   O   R   T   I   N   G       A   L   G   O   R   I   T   H   M   S

                    Press Any Key to Continue
```

```
                    H   E   A   P       S   O   R   T

                    Generating Array Randomly:

39 73 99 49 96 91 93 50 69 8 21 81 66 5 82 70 45 35 60 97 59 58 74 27 26 7 43 66 41 70 84 79 95 34 79 90 76 23 91 44 82 63 77 48 20 58 17 64 93 28
60 3 86 85 29 11 91 71 76 84 92 11 14 86 96 93 27 24 15 69 19 97 32 95 96 51 53 12 66 97 40 77 99 77 61 27 87 52 49 14 35 92 24 48 78 71 92 56 94 7

                    S e r i a l  E x e c u t i o n

                    Sorted Array:     3 5 7 7 8 11 11 12 14 14
                    Time Taken:       0.000032 seconds.

                    P t h r e a d s

                    Sorted Array:     3 5 7 7 8 11 11 12 14 14
                    Time Taken:       0.000250 seconds.

                    O p e n M P

                    Sorted Array:     3 5 7 7 8 11 11 12 14 14
                    Time Taken:       0.001072 seconds.
k200406_fatima@Fatima-VirtualBox:~/Desktop/proj/final$
```

```
k200406_fatima@Fatima-VirtualBox:~/Desktop/proj/final$ gcc -o c Count.c -lpthread -fopenmp
k200406_fatima@Fatima-VirtualBox:~/Desktop/proj/final$ ./c

                    C   O   U   N   T       S   O   R   T

                    Generating Array Randomly:

73 22 45 84 15 9 84 78 6 58 80 35 19 89 8 44 44 11 4 12 99 45 39 75 63 1 52 88 41 79 45 13 53 90 49 67 98 84 45 56 93 76 90 12 64 97 7 8 59 63
71 58 7 9 84 21 10 88 60 2 66 5 67 70 94 15 37 43 50 33 98 42 8 39 5 72 88 12 31 46 26 53 55 84 14 91 5 75 78 64 76 95 20 94 65 13 60 1 8 9

                    S e r i a l  E x e c u t i o n

                    Sorted Array:     1 1 2 4 5 5 5 6 7 7
                    Time Taken:       0.000010 seconds.

                    P t h r e a d s

                    Sorted Array:     1 1 2 4 5 5 5 6 7 7
                    Time Taken:       0.000184 seconds.

                    O p e n M P

                    Sorted Array:     1 1 2 4 5 5 5 6 7 7
                    Time Taken:       0.000603 seconds.
k200406_fatima@Fatima-VirtualBox:~/Desktop/proj/final$
```

```
k200406_fatima@Fatima-VirtualBox:~/Desktop/proj/final$ gcc -o r Radix.c -lpthread -fopenmp
k200406_fatima@Fatima-VirtualBox:~/Desktop/proj/final$ ./r

                    R   A   D   I   X       S   O   R   T

                    Generating Array Randomly:

63 97 95 40 100 47 31 14 11 86 67 74 71 74 89 28 82 11 3 61 31 88 100 5 98 51 34 75 87 80 56 1 28 50 40 28 49 22 41 59 59 7 33 81 32 73 9 14 35 11
26 66 51 77 22 100 27 8 74 65 39 81 65 66 83 56 93 31 77 85 89 88 92 73 68 23 46 28 88 80 91 66 97 93 42 19 92 69 78 66 33 16 98 50 81 80 5 74 10 82

                    S e r i a l  E x e c u t i o n

                    Sorted Array:     1 3 5 5 7 8 9 10 11 11
                    Time Taken:       0.000031 seconds.

                    P t h r e a d s

                    Sorted Array:     1 3 5 5 7 8 9 10 11 11
                    Time Taken:       0.000116 seconds.

                    O p e n M P

                    Sorted Array:     1 3 5 5 7 8 9 10 11 11
                    Time Taken:       0.000426 seconds.
k200406_fatima@Fatima-VirtualBox:~/Desktop/proj/final$
```

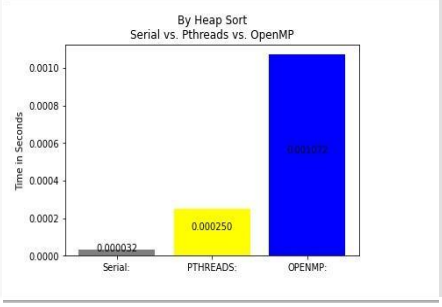
RESULTS AND DISCUSSION

```
from google.colab import files
data_to_load = files.upload()
```

Browse... HeapTime.txt

HeapTime.txt(text/plain) - 52 bytes, last modified: n/a - 100% done
Saving HeapTime.txt to HeapTime.txt

```
import matplotlib.pyplot as plt

file = open("HeapTime.txt")
text = []
for line in file:
    text.append(line)

x = [text[0], text[2], text[4]]
y = [float(text[1]), float(text[3]), float(text[5])]

x_Pos = [i for i, _ in enumerate(x)]

fig, ax = plt.subplots()
rects1 = ax.bar(x_Pos, y, color=('grey', 'yellow', 'blue'))
plt.ylabel("Time in Seconds")
plt.title("By Heap Sort\nSerial vs. Pthreads vs. OpenMP")
plt.xticks(x_Pos, x)

def autolabel(rects):

    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 0.5*height,
                '%f' % float(height),
                ha='center', va='bottom')

autolabel(rects1)

plt.show()
```

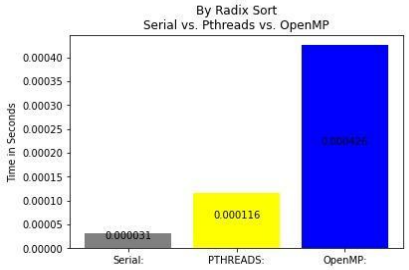



```
from google.colab import files
data_to_load = files.upload()
```

Browse... RadixTime.txt

RadixTime.txt(text/plain) - 52 bytes, last modified: n/a - 100% done
Saving RadixTime.txt to RadixTime.txt

```
import matplotlib.pyplot as plt

file = open("RadixTime.txt")
text = []
for line in file:
    text.append(line)

x = [text[0], text[2], text[4]]
y = [float(text[1]), float(text[3]), float(text[5])]

x_Pos = [i for i, _ in enumerate(x)]

fig, ax = plt.subplots()
rects1 = ax.bar(x_Pos, y, color=('grey', 'yellow', 'blue'))
plt.ylabel("Time in Seconds")
plt.title("By Radix Sort\nSerial vs. Pthreads vs. OpenMP")
plt.xticks(x_Pos, x)

def autolabel(rects):

    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 0.5*height,
                '%f' % float(height),
                ha='center', va='bottom')

autolabel(rects1)

plt.show()
```

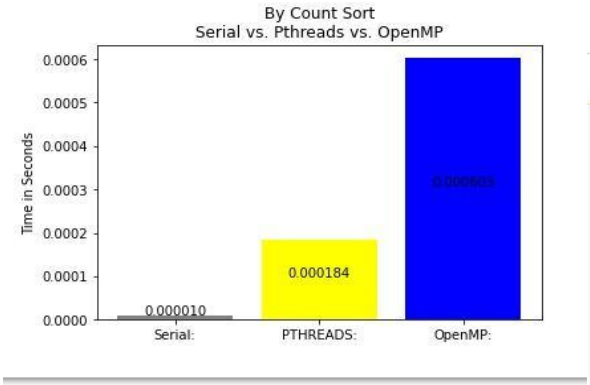



```
from google.colab import files
data_to_load = files.upload()
```

Browse... CountTime.txt

CountTime.txt(text/plain) - 52 bytes, last modified: n/a - 100% done
Saving CountTime.txt to CountTime.txt

```
import matplotlib.pyplot as plt

file = open("CountTime.txt")
text = []
for line in file:
    text.append(line)

x = [text[0], text[2], text[4]]
y = [float(text[1]), float(text[3]), float(text[5])]

x_Pos = [i for i, _ in enumerate(x)]

fig, ax = plt.subplots()
rects1 = ax.bar(x_Pos, y, color=('grey', 'yellow', 'blue'))
plt.ylabel("Time in Seconds")
plt.title("By Count Sort\nSerial vs. Pthreads vs. OpenMP")
plt.xticks(x_Pos, x)

def autolabel(rects):

    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 0.5*height,
                '%f' % float(height),
                ha='center', va='bottom')

autolabel(rects1)

plt.show()
```



CountTime.txt

```
1 Serial:
2 0.000010
3 PTHREADS:
4 0.000184
5 OpenMP:
6 0.000603
```

HeapTime.txt

```
1 Serial:
2 0.000032
3 PTHREADS:
4 0.000250
5 OPENMP:
6 0.001072
```