

DESIGN AND ANALYSIS OF ALGORITHMS

EXPERIMENT 2

ANUSHKA ACHARYA

CSE DS D1 BATCH

UID: 2021700001

AIM: EXPERIMENT BASED ON DIVIDE AND CONQUER APPROACH

THEORY:

Merge sort is a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.

In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted. Merge sort is a popular choice for sorting large datasets because it is relatively efficient and easy to implement. It is often used in conjunction with other algorithms, such as quicksort, to improve the overall performance of a sorting routine.

<https://www.geeksforgeeks.org/merge-sort/>

QuickSort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

- Always pick the first element as a pivot.
- Always pick the last element as a pivot (implemented below)
- Pick a random element as a pivot.
- Pick median as the pivot.

The key process in quickSort is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

<https://www.geeksforgeeks.org/quick-sort/>

CODE:

MERGESORT:

```
#include<stdlib.h>
#include<stdio.h>
#include <time.h>
```

```
void merge(int a[], int l, int mid, int r){
    int n1 = mid-l+1;
    int n2 = r-mid;
    int i,j,k;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = a[l + i];
```

```

for (j = 0; j < n2; j++)
R[j] = a[mid+ 1+ j];
i = 0;
j = 0;
k = 1;
while (i < n1 && j < n2){
if (L[i] <= R[j]){
a[k] = L[i];
i++;
}
else{
a[k] = R[j];
j++;
}
k++;
}
while (i < n1){
a[k] = L[i];
i++;
k++;
}
while (j < n2){
a[k] = R[j];
j++;
k++;
}
}
void mergeSort(int a[], int l, int r){
if (l < r){
int mid= l+(r-l)/2;
mergeSort(a, l, mid);
mergeSort(a, mid+1, r);
merge(a, l, mid, r);
}
}
int main(){
int size = 100;
FILE *fp;
fp = fopen("anu.txt", "r");
while(size<=100000){
double time_taken=0.0;
clock_t begin=clock();
int a[size];
for(int j=0;j<size;j++){
fscanf(fp,"%d",&a[j]);
}
mergeSort(a, 0, size - 1);
for(int test1=1;test1<size;test1++){
printf("%d\t",a[test1]);
}
printf("\n");
size=size+100;
fseek(fp,0,SEEK_SET);
clock_t end=clock();
time_taken = (double)(end-begin)/CLOCKS_PER_SEC;

```

```

printf("Runtime: %f\n",time_taken);
}
return 0;
}

```

QUICKSORT:

```

#include<stdlib.h>
#include<stdio.h>
#include <time.h>

```

```

int partition (int a[], int start, int end)
{
    int pivot = a[end];
    int i = (start - 1);

    for (int j = start; j <= end - 1; j++)
    {
        if (a[j] < pivot)
        {
            i++;
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    int t = a[i+1];
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}

```

```

void quickSort(int a[], int start, int end) {
    if (start < end) {
        int p = partition(a, start, end);
        quickSort(a, start, p - 1);
        quickSort(a, p + 1, end);
    }
}

int main(){
int size = 100;
FILE *fp;
fp = fopen("anu.txt","r");
while(size<=100000){
double time_taken=0.0;
clock_t begin=clock();
int a[size];
for(int j=0;j<size;j++){
fscanf(fp,"%d",&a[j]);
}
quickSort(a, 0, size - 1);
for(int test1=1;test1<size;test1++){
printf("%d\t",a[test1]);
}
printf("\n");
size=size+100;
}

```

```

fseek(fp,0,SEEK_SET);
clock_t end=clock();
time_taken = (double)(end-begin)/CLOCKS_PER_SEC;
printf("Runtime: %fn",time_taken);
}
return 0;
}

```

OUTPUT:

Mergesort output for first three blocks:

```

students@lenovo-ThinkCentre-neo-50s-Gen-3: ~/Desktop
0.009227
0.000720
0.000408
0.007991
0.008520
0.008504
0.007638
0.008834
students@lenovo-ThinkCentre-neo-50s-Gen-3:~/Desktop$ ^C
students@lenovo-ThinkCentre-neo-50s-Gen-3:~/Desktop$ gcc merge.c
students@lenovo-ThinkCentre-neo-50s-Gen-3:~/Desktop$ ./a.out
3624 4364 4852 4938 5122 5326 6902 7644 9667 10188 10207 12945 15492 15501 16076 16293 17561 20105 22285 23139 23610
25940 25960 27730 28787 30102 32647 32766 36553 36560 36606 38501 38578 38765 41524 41925 42028 42790 43433 43945 44508 44536 45014
46839 49658 50209 52352 53755 54023 54698 55569 57230 57426 57843 58323 59995 61807 62062 64614 67560 67570 69034 70464 71995
72415 74771 75058 75832 77542 77677 78716 78734 79222 80601 80891 85671 86007 86138 87153 88224 88421 88658 90808 90814 90827
90961 91413 91687 91844 91908 92297 92501 92616 93293 95323 96345 97492 97828 98531 99414
Runtime: 0.000102
1207 2361 3461 3624 3856 3939 4364 4454 4567 4852 4938 5122 5326 5397 5442 5705 6902 7211 7215 7278 7644
8726 9667 9973 10188 10207 11669 11881 12554 12945 14053 15492 16076 16293 17561 20105 20275 21848 22285 22783 23139
23610 25103 25940 25960 27730 28787 28961 29336 29355 30049 30096 30102 31556 32405 32647 32694 32766 34830 36553 36560 36606
36884 38501 38578 38765 39116 40787 41075 41232 41524 41925 42028 42448 42790 43433 43945 44508 44536 45014 46184 46502 46583
46839 48536 49639 49658 50209 50760 50867 52352 52525 53609 53727 53755 54023 54698 54704 55500 55569 55614 56381 57230 57426
57572 57843 57847 58034 58070 58323 58542 58692 59202 59995 60511 60791 61807 62062 62304 62646 64614 67560 67570 69034 69108
70417 70331 70464 71389 71995 72415 72554 73422 74434 74771 74846 75058 75094 75832 76491 77263 77542 77644 77677 77823
77990 78716 78734 79222 80601 80891 81082 81740 82149 82527 84361 84754 85002 85064 85108 85430 85671 85764 85919
86409 86729 87153 88123 88224 88421 88658 90808 90814 90827 90961 91413 91687 91844 91908 92297 92501 92616 93293 94305
95323 96345 96944 97492 97828 97916 98505 98531 99414
Runtime: 0.000170
896 987 989 1072 1207 2361 3243 3461 3624 3856 3871 3896 3930 3939 4364 4454 4567 4852 4938 5122 5326
5397 5442 5705 5971 6259 6902 7211 7215 7278 7644 8726 8858 9000 9041 9667 9838 9973 10068 10188 10207 10947
11368 11497 11669 11881 12554 12945 13112 13128 13357 14053 14218 14817 15492 15501 16076 16293 16383 17227 17561 17981 19311
20275 20275 21848 22007 22190 22285 22672 22783 23139 23610 24067 25140 25163 25764 25940 25960 26602 27205 27730 28787 28961
29190 29336 29355 29600 29951 30049 30096 30102 31111 31469 31556 32405 32647 32694 32766 33783 34607 34830 35078 35138 35833
35836 36240 36553 36560 36606 36884 37366 38353 38501 38578 38765 39116 40194 40787 41075 41232 41297 41524 41925 42028 42448
42790 43433 43945 44508 44536 45014 46184 46502 46583 46839 48536 49639 49658 50209 50760 50867 51272 52352 52525 53609 53727 53755 54023 54698 54704 55500 55569 55614 56381 57230 57426 57572
57843 57847 58034 58070 58323 58542 58692 59202 59954 59995 60511 60522 60791 61807 61927 62062 62304 62646 63902 64095
64597 64614 67387 67560 67570 67666 69034 69108 69411 70147 70331 70343 70464 70499 70848 71389 71995 72382 72415 72554 73422
73522 74434 74727 74771 74846 75058 75094 75832 76491 77263 77542 77644 77677 77823 77990 78716 78734 79222 79290 79650 80624
80836 80601 80891 81021 81082 81740 81945 82149 82527 83062 84251 84361 84754 85002 85064 85108 85430 85671 85764 85919
86007 86103 86138 86409 86513 86729 87153 87231 88123 88224 88421 88658 89079 90286 90808 90814 90827 90961 91413 91433 91687
91844 91896 91908 92297 92501 92598 92616 93293 93405 94182 94822 95323 95405 95600 96345 96944 97404 97492 97828 97916 98505
98531 99121 99145 99414 99770
Runtime: 0.000307
students@lenovo-ThinkCentre-neo-50s-Gen-3:~/Desktop$

```

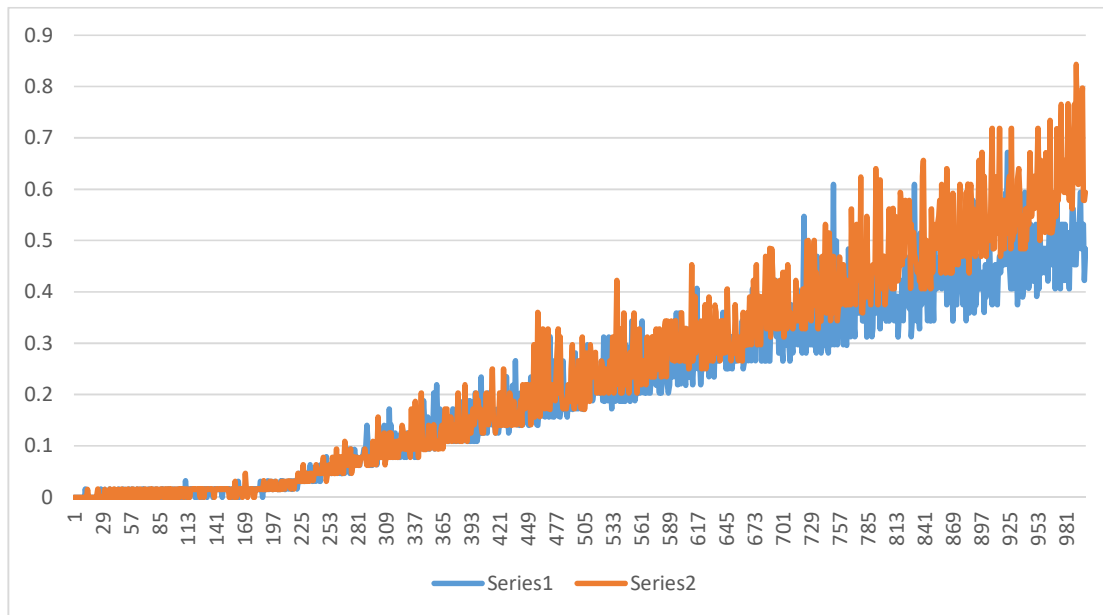
Quicksort output for first three blocks:

```

C:\Users\user\Desktop\DA>quicksort.o
3624 4364 4852 4938 5122 5326 6902 7644 9667 10188 10207 12945 15492 15501 16076
16293 17561 20105 22285 23139 23610 25940 25960 27730 28787 30102 32647 32766 36553
36560 36606 38501 38578 38765 39116 40787 41075 41232 41524 41925 42028 42448 42790 43433 43945 44508 44536 45014 46184 46502 46583 46839
49658 50209 52352 53755 54023 54698 55569 57230 57426 57843 58323 59995 61807 62062 64614 67560 67570 69034 70464 71995
72415 74771 75058 75832 77542 77677 78716 78734 79222 80601 80891 85671 86007 86138 87153 88224 88421 88658 90808 90814 90827 90961
91413 91687 91844 91908 92297 92501 92616 93293 95323 96345 97492 97828 98531 99414
Runtime: 0.015009
1207 2361 3461 3624 3856 3939 4364 4454 4567 4852 4938 5122 5326 5397 5442
5705 6902 7211 7215 7278 7644 8726 9667 9973 10188 10207 11669 11881 12554
12945 14053 15492 15501 16076 16293 17561 20105 20275 21848 22285 22783 23139 23610
25163 25764 25940 25960 27730 28787 28961 29336 29355 30049 30096 30102 31556 32405 32647
32694 32766 34830 36553 36560 36606 36884 38501 38578 38765 39116 40787 41075 41232
41524 41925 42028 42448 42790 43433 43945 44508 44536 45014 46184 46502 46583 46839
48536 49639 49658 50209 50760 50867 52352 52525 53609 53727 53755 54023 54698 54704
55500 55569 55614 56381 57230 57426 57572 57843 57847 58034 58070 58323 58542 58692
59202 59995 60511 60791 61807 62062 62304 62646 64614 67560 67570 69034 69108 70147
70331 70343 70464 71389 71995 72415 72554 73422 74434 74771 74846 75058 75094 75832
76491 77263 77542 77644 77677 77823 77990 78716 78734 79222 80601 80891 81082
81740 82149 82527 84361 84754 85002 85430 85671 85764 85919 86007 86103 86138 86409
86729 87153 88123 88224 88421 88658 90808 90814 90827 90961 91413 91687 91844 91896
91908 92297 92501 92616 93293 94822 95323 96345 96944 97492 97828 97916 98505 98531
99145 99414
Runtime: 0.031000
896 987 989 1072 1207 2361 3243 3461 3624 3856 3871 3896 3930 3939 4364
4454 4567 4852 4938 5122 5326 5397 5442 5705 5971 6259 6902 7211 7215
7278 7644 8726 8858 9000 9041 9667 9838 9973 10068 10188 10207 10947 11368
11497 11669 11881 12554 12945 13112 13128 13357 14053 14218 14817 15492 15501 16076
16293 16383 17227 17561 17981 19311 20105 20275 21848 22007 22190 22285 22672 22783
23139 23610 24067 25140 25163 25764 25940 25960 26602 27205 27730 28787 28961 29190
29336 29355 29600 29951 30049 30096 30102 31111 31469 31556 32405 32647 32694 32766
33783 34607 34830 35078 35138 35833 36240 36553 36560 36606 36884 37366 38353
38501 38578 38765 39116 40194 40787 41075 41232 41297 41524 41925 42028 42448 42790
42900 43433 43945 44508 44536 45014 46184 46384 46502 46583 46614 46839 47101 47345
47900 48536 49639 49658 50209 50760 50867 51272 52352 52525 53609 53727 53755
53927 54823 54227 54698 54704 55500 55569 55596 55614 56381 57230 57426 57572 57572
57843 57847 58034 58070 58323 58542 58692 59202 59954 59995 60511 60522 60791 61807 61927 62062 62304 62646 63902 64095
61927 62062 62304 62646 63902 64095 64597 64614 67387 67560 67570 67666 69034 69108 69411 70147 70331 70343 70464 70499 70848 71389 71995 72382 72415 72554 73422 73522
74434 74727 74771 74846 75058 75094 75832 76491 77263 77542 77644 77677 77823 77990 78716 78734 79222 79290 79650 80624
80836 80601 80891 81021 81082 81740 81945 82149 82527 83062 84251 84361 84754 85002 85064 85108 85430 85671 85764 85919
86007 86103 86138 86409 86513 86729 87153 88123 88224 88421 88658 90808 90814 90827 90961 91413 91433 91687
91844 91896 91908 92297 92501 92598 92616 93293 93405 94182 94822 95323 95405 95600 96345 96944 97404 97492 97828 97916 98505
98531 99121 99145 99414 99770

```

GRAPH:



OBSERVATION:

In this graph, the blue line represents merge sort whereas the orange line represents quicksort algorithm. It can be seen clearly then quicksort has a higher runtime than the merge sort algorithm. This is because in the merge sort, the array is parted into just 2 halves (i.e. $n/2$) whereas in case of quick sort, the array is parted into any ratio. The worst case complexity of quick sort is $O(n^2)$ as there is need of lot of comparisons in the worst condition. Whereas in merge sort, worst case and average case has same complexities $O(n \log n)$. Thus, even though both algorithms work on divide and conquer strategy, merge sort is more efficient of the two when dealing with a huge set of data. Quicksort is usually preferred for small data sets.

CONCLUSION:

From this experiment I learnt how to perform merge sort and quicksort algorithms. I used the sorting techniques on 100000 random numbers and found out that runtime for merge sort is less than quicksort which helped me analyse that merge sort is the algorithm more efficient between the two, when working on a large set of data.