**LAPORAN RESMI**

**SORTING**



**Umi Sa’adah S.Kom., M.Kom.**

Disusun Oleh :

Nama : Daniar Oktavian Dwiputra

Kelas : 1 / D4 Teknik Informatika A

NRP : 3122600030

1. Listing Program

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<time.h>

#define MAX 100000

void selection(int [], int);

void insertion(int [], int);

void bubble(int [], int);

void shell(int [], int);

void mergeSortRekursif(int [], int , int);

void merge(int [], int , int, int);

void quickSort(int [], int, int);

int partition(int [], int, int);

void tampil(int [], int);

void tukar(int \*, int \*);

void generate(int [], int);

void waktu(clock\_t, clock\_t, int);

void menu();

int main()

{

menu();

return 0;

}

void menu()

{

int n, jwb, arr[MAX], backup[MAX];

clock\_t start, end;

printf("Berapa jumlah data (maks 100000) ? ");

scanf("%d", &n);

generate(arr, n);

do

{

memcpy(backup, arr, sizeof(int) \* n);

printf("\nMENU METODE SORTING\n");

printf("1. Insertion\n");

printf("2. Selection\n");

printf("3. Bubble\n");

printf("4. Shell\n");

printf("5. Merge\n");

printf("6. Quick\n");

printf("7. Keluar\n");

printf("Pilihan Anda : ");

scanf("%d", &jwb);

switch (jwb)

{

case 1:

insertion(backup, n);

break;

case 2:

selection(backup, n);

break;

case 3:

bubble(backup, n);

break;

case 4:

shell(backup, n);

break;

case 5:

start = clock();

mergeSortRekursif(backup, 0, n);

end = clock();

waktu(start, end, n);

break;

case 6:

srand(time(NULL));

start = clock();

quickSort(backup, 0, n-1);

end = clock();

waktu(start, end, n);

break;

case 7:

printf("\nPROGRAM DIHENTIKAN\n");

exit(0);

break;

default:

printf("\nPilihan Anda Invalid\n");

break;

}

} while (jwb != 7);

}

void insertion(int x[], int n)

{

int i, j, key;

clock\_t start, end;

start = clock();

i = 1;

while(i < n)

{

key = x[i];

j = i - 1;

while (j >= 0 && (x[j] > key))

{

x[j+1] = x[j];

j--;

}

x[j+1] = key;

i++;

}

end = clock();

waktu(start, end, n);

}

void selection(int x[], int n)

{

int i, j, min;

clock\_t start, end;

start = clock();

while(i < n)

{

min = i;

j = i + 1;

while(j < n)

{

if(x[j] < x[min])

{

min = j;

}

j++;

}

tukar(&x[i], &x[min]);

i++;

}

end = clock();

waktu(start, end, n);

}

void bubble(int x[], int n)

{

int i, j, did\_swap;

clock\_t start, end;

did\_swap = 1;

start = clock();

for(i = 0; i < n-1; i++)

{

if (did\_swap)

{

did\_swap = 0;

for(j = 0; j < (n - i - 1); j++)

{

if (x[j] > x[j+1])

{

tukar(&x[j], &x[j+1]);

did\_swap = 1;

}

}

}

}

end = clock();

waktu(start, end, n);

}

void shell(int x[], int n)

{

int jarak = n / 2;

int i, did\_swap;

clock\_t start, end;

start = clock();

while (jarak > 0)

{

did\_swap = 1;

while (did\_swap == 1)

{

did\_swap = 0;

for (i = 0; i < n - jarak; i++)

{

if (x[i] > x[i + jarak])

{

tukar(&x[i], &x[i + jarak]);

did\_swap = 1;

}

}

}

jarak = jarak / 2;

}

end = clock();

waktu(start, end, n);

}

void mergeSortRekursif(int data[], int l, int r)

{

int med;

if(l < r)

{

med = (l+r) / 2;

mergeSortRekursif(data, l, med);

mergeSortRekursif(data, med+1, r);

merge(data, l, med, r);

}

}

void merge(int data[], int l , int m, int r)

{

int i, j, ki1, ki2, ka1, ka2;

int hasil [MAX];

ki1 = l;

ka1 = m;

ki2 = m+1;

ka2 = r;

i = l;

while (ki1 <= ka1 && ki2 <= ka2)

{

if(data[ki1] <= data[ki2])

{

hasil[i] = data[ki1];

ki1++;

}

else

{

hasil[i] = data[ki2];

ki2++;

}

i++;

}

while (ki1 <= ka1)

{

hasil[i] = data[ki1];

ki1++;

i++;

}

while (ki2 <= ka2)

{

hasil[i] = data[ki2];

ki2++;

i++;

}

j = l;

while (j <= r)

{

data[j] = hasil[j];

j++;

}

}

void quickSort(int A[], int p, int r)

{

int q;

if (p < r)

{

q = partition(A, p, r);

quickSort(A, p, q-1);

quickSort(A, q+1, r);

}

}

int partition(int A[], int p , int r)

{

int i, j, x;

x = A[r];

i = p - 1;

for (j = p; j < r; j++)

{

if (A[j] <= x)

{

i++;

tukar(&A[i], &A[j]);

}

}

tukar(&A[i+1], &A[r]);

return (i+1);

}

void tukar(int \*a, int \*b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

void tampil(int A[], int n)

{

int i;

for (i = 0; i < n; i++)

{

printf("%d ", A[i]);

}

printf("\n");

}

void generate(int x[], int n)

{

int i;

srand(time(NULL));

for(i = 0; i<n; i++)

{

x[i] = rand()/1000;

}

}

void waktu(clock\_t start, clock\_t end, int n)

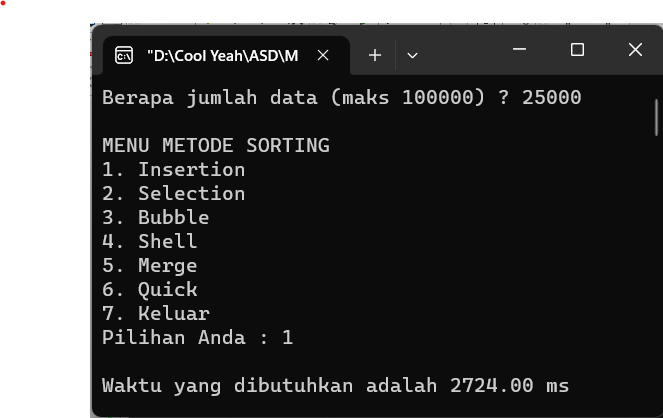
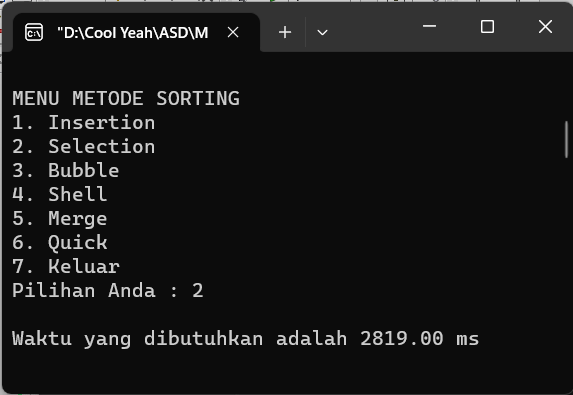
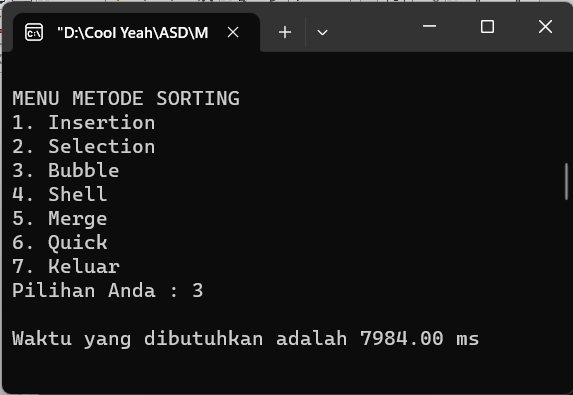
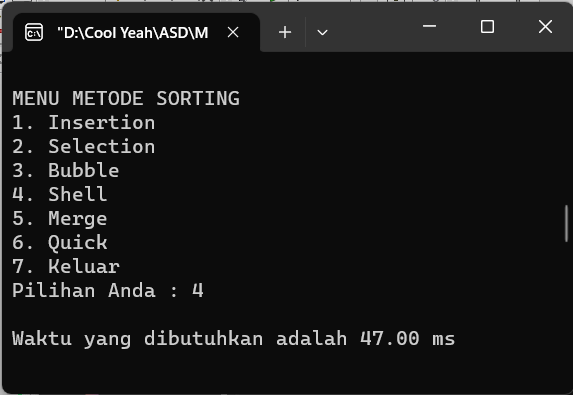
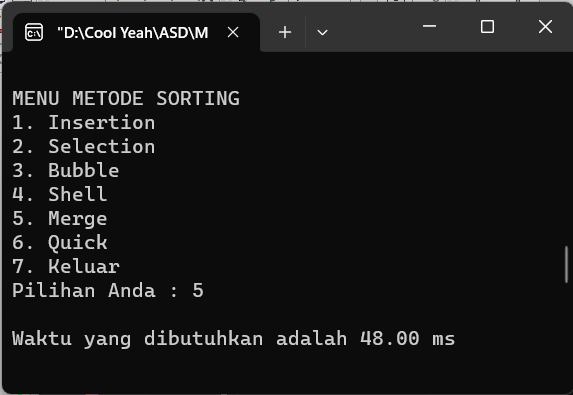
{

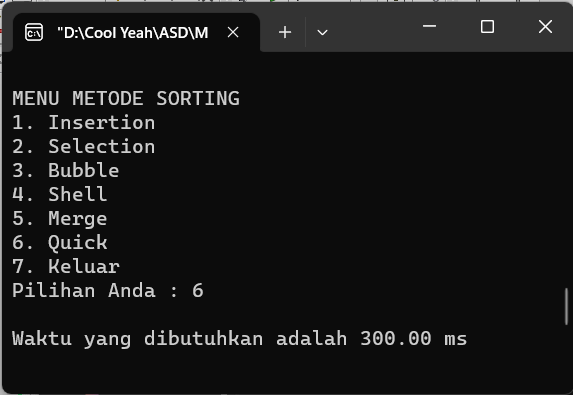
double cpu\_time\_used;

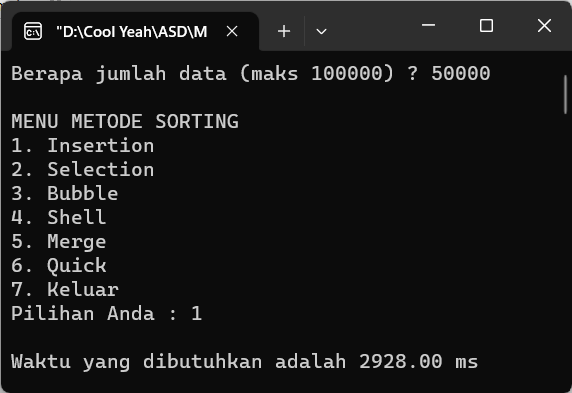
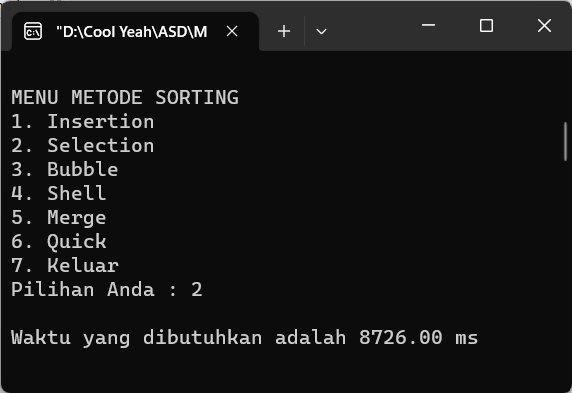
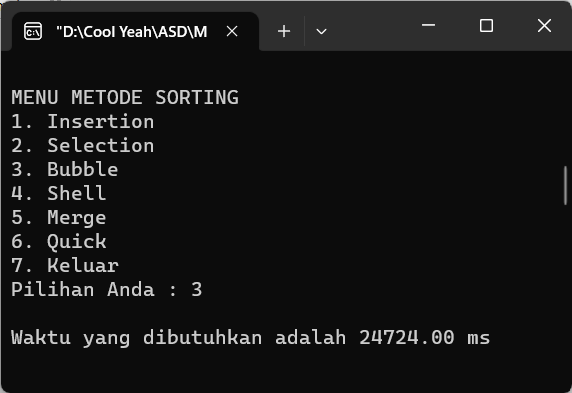
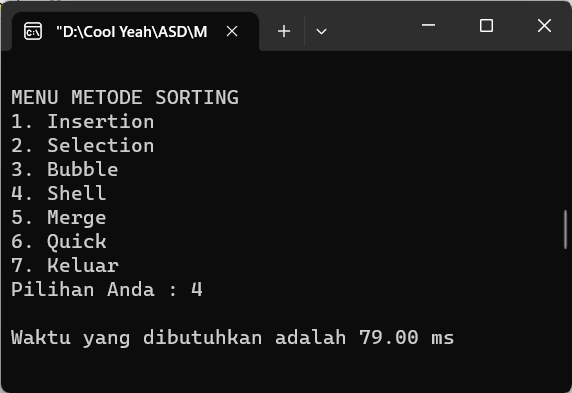
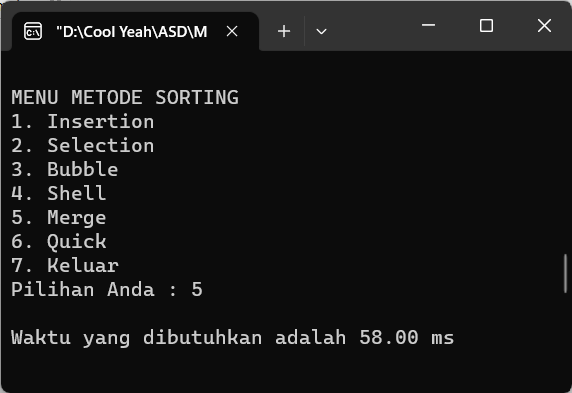
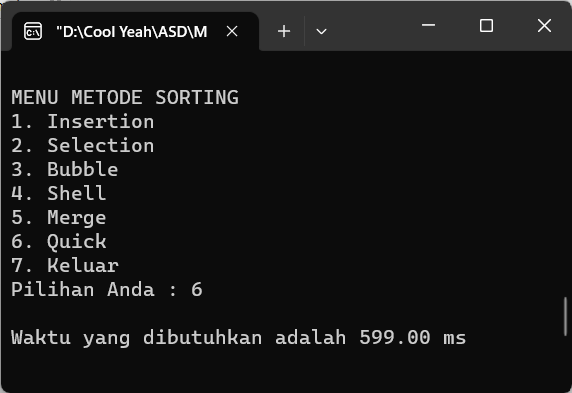
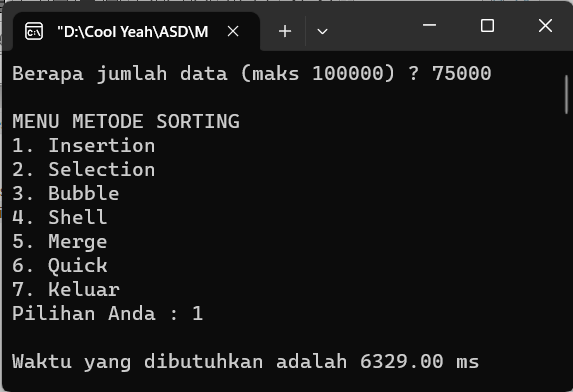
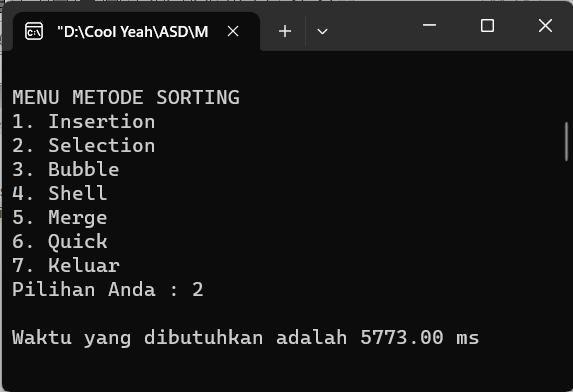
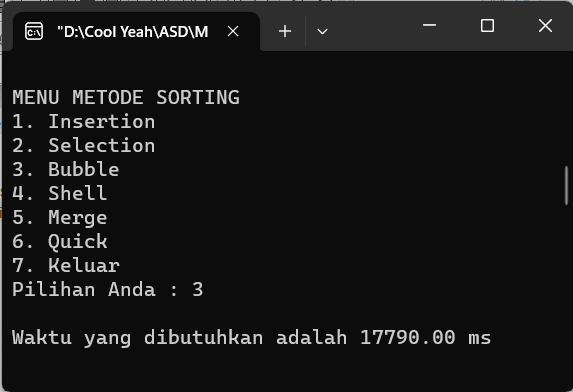
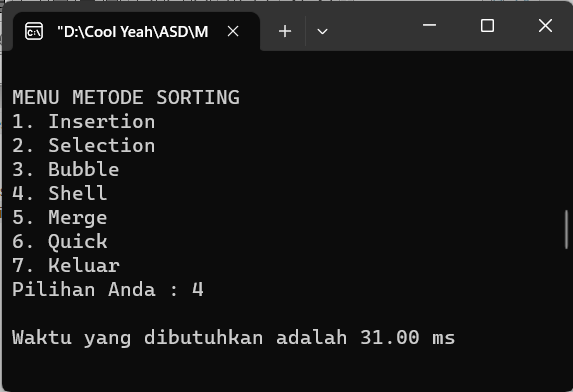
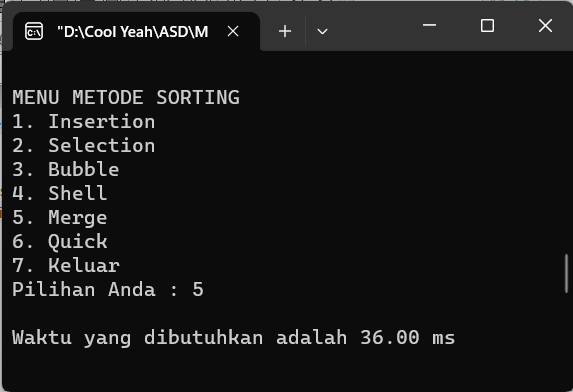
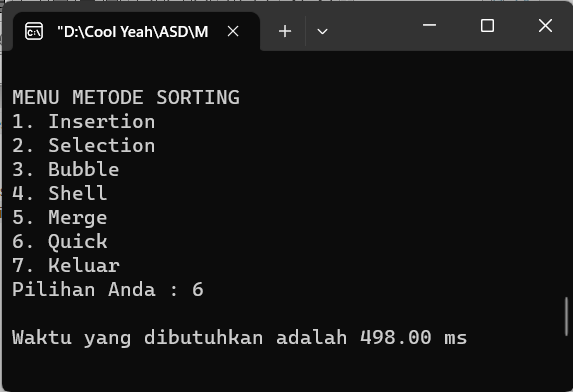
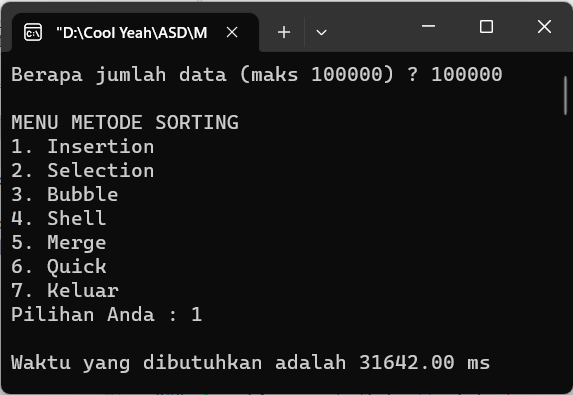
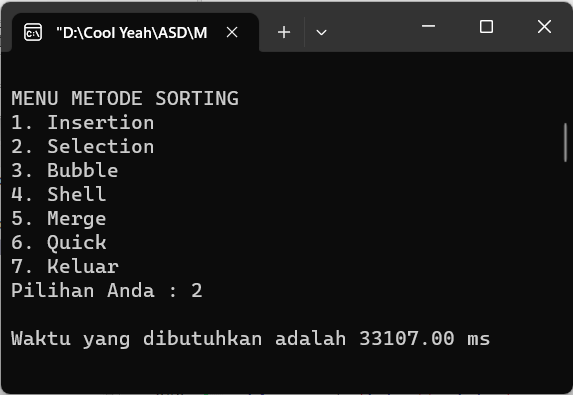
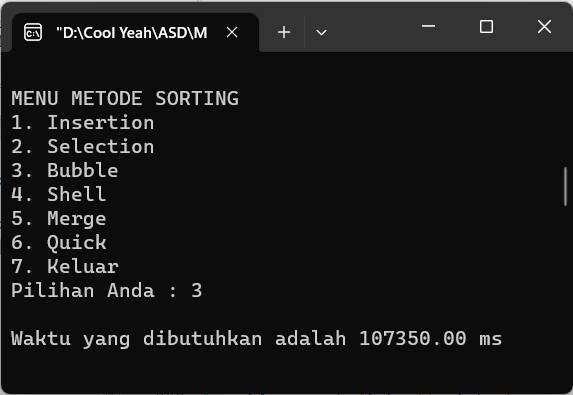
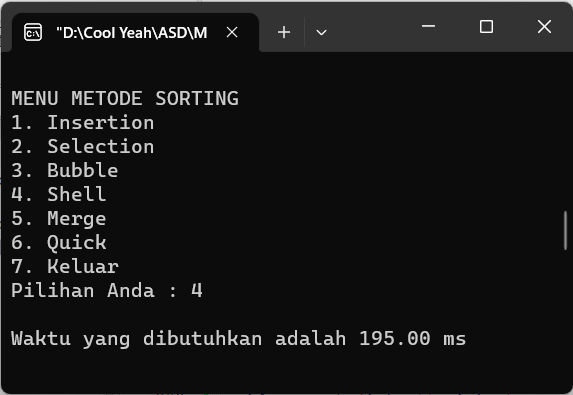
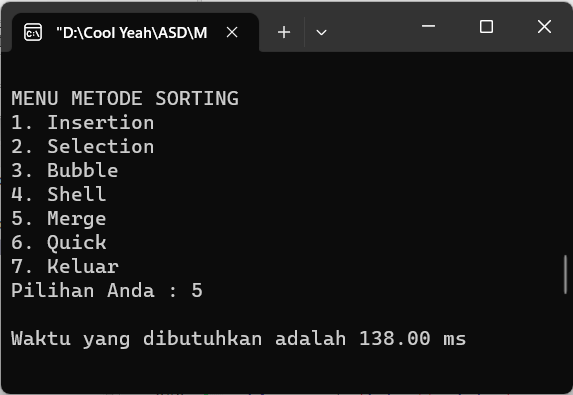
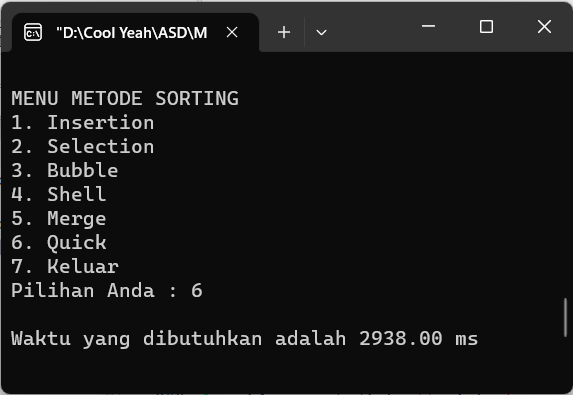
cpu\_time\_used = ((double)end - start) / CLOCKS\_PER\_SEC;

printf("\nWaktu yang dibutuhkan adalah %.2f ms\n", cpu\_time\_used \* 1000);

}

1. Implementasi 6 Metode Sorting
2. 25000 data
3. Insertion
4. Selection
5. Bubble
6. Shell
7. Merge
8. Quick



1. 50000 data
2. Insertion
3. Selection
4. Bubble
5. Shell
6. Merge
7. Quick
8. 75000 data
9. Insertion
10. Selection
11. Bubble
12. Shell
13. Merge
14. Quick
15. 100000 data
16. Insertion
17. Selection
18. Bubble
19. Shell
20. Merge
21. Quick
22. Perbandingan performa masing-masing metode
23. 25000 Data
24. 50000 Data
25. 75000 Data
26. 100000 Data
27. Analisa dan Kesimpulan

Dari hasil percobaan yang telah dilakukan dan dapat dilihat dari data di atas, metode sorting yang tercepat adalah shell dan yang paling lambat adalah Bubble sort.