(2 sorting methods)

1. Direct Include

```
//Code by Xumingchuan from group 2

//matrix dimensions take from 20 to 4860, with 3 times proportion between dimensions
//Update proportion - 20%


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

//different dimensions of matrix
#define nDim 20//60、180、540、1620、4860

int Arr[nDim];

//initialization
void RndArr(){
   for(int i=0;i<nDim;i++)
      Arr[i]=rand()%1000;
}

//Update 20percent data front
void RndArrFront(){
      int nDimFront=nDim/5;
      for(int i=0;i<nDimFront;i++)
         Arr[i]=rand()%1000;
}

//Update 20percent data end
void RndArrEnd(){
      int nDimEnd=nDim-nDim/5;
      for(int i=nDimEnd;i<nDim;i++)
         Arr[i]=rand()%1000;
}

void PrintArr(){
   for(int i=0;i<nDim;i++)
      printf("%d ",Arr[i]);
}
```

```
int nCount;

void Exchange(int i,int j){
   int Tmp=Arr[i];
   Arr[i]=Arr[j];
   Arr[j]=Tmp;
   nCount += 3;
}

//move 1 position towards the back
void shift(int i,int j){
   int Tmp=Arr[j];
   if(j>i) for(int k=j;k>=i+1;k--) Arr[k]=Arr[k-1];
   else       for(int k=j;k<=i-1;k++) Arr[k]=Arr[k+1];
   Arr[i]=Tmp;
   nCount+= abs(i-j)+2;
}

//find the first number which bigger than key
int LinSearch(int key,int iMax){
   for(int i=0;i<=iMax;i++)
      if(Arr[i]>=key)
         return i;
   return -1;
}

void SortingDirectInclude(){
   for(int i=1;i<nDim;i++){
      int j=LinSearch(Arr[i],i-1);
      if(j>=0) shift(j,i);
   }
}

void UpdateFront(){
   printf("Update 20percent data front:\n");
   RndArrFront();
   PrintArr();
   nCount=0;
   SortingDirectInclude();
   printf("\nAfter Direct Include with 20percent new data front:\n");
   PrintArr();
   printf("\n");
   printf("nCount=%d",nCount);
```

```
    printf("\n");
    printf("\n");
}

void UpdateEnd(){
    printf("Update 20percent data end:\n");
    RndArrEnd();
    PrintArr();
    nCount=0;
    SortingDirectInclude();
    printf("\nAfter Direct Include with 10percent new data End:\n");
    PrintArr();
    printf("\n");
    printf("nCount=%d",nCount);
    printf("\n");
}


int main()
{
    printf("nDim=%d\n",nDim);
    RndArr();
    printf("The original sequence:\n");
    PrintArr();
    nCount=0;
    SortingDirectInclude();
    printf("\nAfter Direct Include:\n");
    PrintArr();
    printf("\n");
    printf("nCount=%d",nCount);
    printf("\n");
    printf("\n");

    char Judge;
    printf("Type in F for UpdateFront\n");
    printf("Type in E for UpdateEnd\n");
    scanf("%c",&Judge);
    if(Judge=='F')
        UpdateFront();
    else if(Judge=='E')
        UpdateEnd();

    return 0;
}
```

размерность	пересылок (после обычной генерации)	пересылок (после замены первых 20%)	отношения числа обменов с модификацией первых 20% элементов к числу обменов без модификации	пересылок (после замены последних 20%)
20	93	37	0.398	40
60	923	274	0.297	389
180	9060	3053	0.337	2196
540	78761	25049	0.318	19222
1620	650136	186118	0.286	211571
4860	5969429	1713104	0.287	1824433

2. Heap Sort

```
//Code by Xumingchuan from group 2

//matrix dimensions take from 25 to 800, with 2 times proportion between dimensions
//Update proportion - 10%

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

//From Arr[0] start storing numbers
#define nDim 10

int nCount;
int Arr[nDim];

//initialization
void RndArr(){
   for(int i=0;i<nDim;i++)
       Arr[i]=rand()%1000;
}

//Update 10percent data front
void RndArrFront(){
       int nDimFront=nDim/10;
       for(int i=0;i<nDimFront;i++)
          Arr[i]=rand()%1000;
}
```

```
//Update 10percent data end
void RndArrEnd(){
	int nDimEnd=nDim-nDim/10;
	for(int i=nDimEnd;i<nDim;i++)
	   Arr[i]=rand()%1000;
}

void Exchange(int i,int j){
   int tmp=Arr[i];
   Arr[i]=Arr[j];
   Arr[j]=tmp;
   nCount += 3;
}

void PrintArr(){
	for(int i=0;i<nDim;i++)
		printf("%d ",Arr[i]);
}

int FindBiggerChild(int Child,int Dim){
	//if left and right child exist
	if(Child+1<Dim && Arr[Child+1]>Arr[Child]) Child++;
	return Child;
}

//sink the element to corresponding position
void Sink(int i,int Dim){
	int LeftChild,BiggerChild;
	//if left child exists
	while(2*i+1<Dim){
		LeftChild = 2*i+1;
		//find the biggest child of node
		BiggerChild=FindBiggerChild(LeftChild,Dim);
		//if node bigger than child, then position don't exchange
		if (Arr[i] > Arr[BiggerChild]) break;
		// if the biggest child bigger than node, then exchange the position
		Exchange(i,BiggerChild);
		//make BiggerChile become new node
		i=BiggerChild;
	}
}

void HeapSort(){
	for (int i=nDim/2;i>=0;i--)
```

```
            Sink(i,nDim);       //build heap:start from Arr[1]
        int Dim=nDim;
        while(Dim>1){
            //exchange the head and last element
            Exchange(0,Dim-1);
            //decrease size of Arr and leave sorted element behind
            Dim-=1;
            Sink(0,Dim-1);
        }
}

void UpdateFront(){
    printf("Update 10percent data front:\n");
    RndArrFront();
    PrintArr();
    nCount=0;
    HeapSort();
    printf("\nAfter Heap Sort with 10percent new data front:\n");
    PrintArr();
    printf("\n");
    printf("nCount=%d",nCount);
    printf("\n");
    printf("\n");
}

void UpdateEnd(){
    printf("Update 10percent data end:\n");
    RndArrEnd();
    PrintArr();
    nCount=0;
    HeapSort();
    printf("\nAfter Heap Sort with 10percent new data End:\n");
    PrintArr();
    printf("\n");
    printf("nCount=%d",nCount);
    printf("\n");
}


int main(){
        printf("nDim=%d\n",nDim);
        RndArr();
        printf("The original sequense:\n");
        PrintArr();
```

```
    printf("\n");
    HeapSort();
    printf("After Heap sort:\n");
    PrintArr();
    printf("\n");
    printf("nCount=%d",nCount);
    printf("\n");
    printf("\n");

    char Judge;
    printf("Type in F for UpdateFront\n");
    printf("Type in E for UpdateEnd\n");
    scanf("%c",&Judge);
    printf("\n");

    if(Judge=='F')
       UpdateFront();
    else if(Judge=='E')
       UpdateEnd();

    return 0;
}
```

размерность	пересылок (после обычной генерации)	пересылок (после замены первых 10%)	отношения числа обменов с модификацией первых 10% элементов к числу обменов без модификации	пересылок (после замены последних 10%)
25	288	303	1.052	300
50	732	741	1.012	780
100	1749	1770	1.012	1875
200	4101	4110	1.002	4389
400	9276	9276	1.000	9981
800	21102	20760	0.984	22386