

ວຽກບ້ານ Algorithm

I. ຄວາມໝາຍຂອງການລຽງລຳດັບຂໍ້ມູນ (sorting)

ການຈັດລຽງລຳດັບໝາຍເຖິງການຈັດລຽງຂໍ້ມູນໃຫ້ລຽງລຳດັບຕາມເງື່ອນໄຂທີ່ກຳນົດໄວ້ (ຈາກຫລາຍໄປຫນ້ອຍ ຫລື ຈາກຫນ້ອຍໄປຫລາຍ) ໃນກໍລະນີຂໍ້ມູນແຕ່ລະ record ມີຫລາຍ field ເຮົາກໍຕ້ອງເລືອກ field ທີ່ສົນໃຈມາຈັດລຽງເຊັ່ນ: ປະຫວັດນັກສຶກສາອາດຈະໃຊ້ຫມາຍເລກໄປຈຳໂຕຂອງນັກສຶກສາໄດ້.

ປະໂຫຍດຂອງການຈັດລຽງຂໍ້ມູນ ແມ່ນຊ່ວຍໃຫ້ການຄົ້ນຫາຂໍ້ມູນການເຂົ້າຫາຂໍ້ມູນໄດ້ສະດວກແລະວອງໄວຫຍິງຂຶ້ນ.

II. ປະເພດຂອງການຈັດລຽງລຳດັບຂໍ້ມູນ (sorting)

ການຈັດລຽງຂໍ້ມູນມີສອງໄປເພດຄື:

- ການຈັດລຽງພາຍໃນ (internal sorting) ແມ່ນການຈັດລຽງຂໍ້ມູນທີ່ເກັບໃນຫນ່ວຍຄວາມຈຳຂອງເຄື່ອງຄອມພິວເຕີ (RAM) ການຈັດລຽງແບບນີ້ຈະຕ້ອງອາໃສເທັກນິກແລະວິທີການຂອງໂຄງສ້າງຂໍ້ມູນເຊັ່ນການໃຊ້ Array ຫລື linked-list ເຂົ້າມາຊ່ວຍ.
- ການຈັດລຽງພາຍນອກ (External sorting) ແມ່ນການຈັດລຽງຂໍ້ມູນໃນສື່ບັນທຶກຂໍ້ມູນເຊັ່ນ: Disk. ໂດຍທົ່ວໄປການຈັດລຽງປະເພດນີ້ມັກໃຊ້ເກັບຂໍ້ມູນທີ່ມີຈຳນວນຫລາຍທີ່ບໍ່ສາມາດເກັບໄວ້ໃນຫນ່ວຍຄວາມຈຳໄດ້ຫມົດເຊິ່ງການຈັດລຽງແບບນີ້ຈະຕ້ອງແບບຂໍ້ມູນອອນເປັນສ່ວນຍ່ອຍໆແລ້ວນຳມາລຽງດ້ວຍການຈັດລຽງພາຍໃນກ່ອນແລ້ວຈຶ່ງນຳແຕ່ລະສ່ວນຍ່ອຍມາລວມກັນ.

III. ວິທີການໃນການລຽງລຳດັບຂໍ້ມູນ (sorting)

1. ການຈັດລຽງລຳດັບຂໍ້ມູນແບບຟອງອາກາດ Bubble sort

ການຈັດລຽງລຳດັບແບບຟອງອາກາດ Bubble sort ຈະເຮັດໂດຍການປຽບທຽບຄ່າຂໍ້ມູນທີ່ຢູ່ຕິດກັນໄປເລື້ອຍໆ ຖ້າໃນກໍລະນີລຽງຂໍ້ມູນຈາກຫນ້ອຍໄປຫລາຍ ແລ້ວຄ່າທີ່ 1 ມີຄ່າຫລາຍກວ່າຄ່າທີ່ 2 ກໍຈະເຮັດການສະລັບຕຳແໜ່ງກັນ. ໂດຍວິທີນີ້ຈະເຮັດໃຫ້ຂໍ້ມູນທີ່ມີຄ່າຫນ້ອຍລອຍຂຶ້ນແບບຟອງອາກາດ ແລະ ຂໍ້ມູນຫນ້ອຍສຸດຈະຢູ່ໃນຕຳແໜ່ງຫນ້າສຸດຂອງຊຸດຂໍ້ມູນເຊິ່ງເອີ້ນວ່າການຈັດລຽງແບບ bubble sort.

ຕົວຢ່າງ: ການລຽງຂໍ້ມູນຈາກຫນ້ອຍໄປຫລາຍ ກຳນົດເລກ 5 ໂຕ ຄື: 5 4 3 2 1

bubble sort ເທື່ອທີ 1

5 4 3 2 1

4 5 3 2 1

4 3 5 2 1

4 3 2 5 1

4 3 2 1 5

bubble sort ເທື່ອທີ 2

4 3 2 1 5

3 4 2 1 5

3 2 4 1 5

3 2 1 4 5

3 2 1 4 5

bubble sort ເທື່ອທີ 3

3 2 1 4 5

2 3 1 4 5

2 1 3 4 5

2 1 3 4 5

2 1 3 4 5

bubble sort ເທື່ອທີ 4

2 1 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

ຜົນຈາກການຈັດລຽງ **bubble sort** ໄດ້: 1 2 3 4 5

ໂປແກຣມ bubble sort ໃນພາສາ c

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

```
int main()
```

```
{
```

```
    int array[100], n, c, d, swap;
```

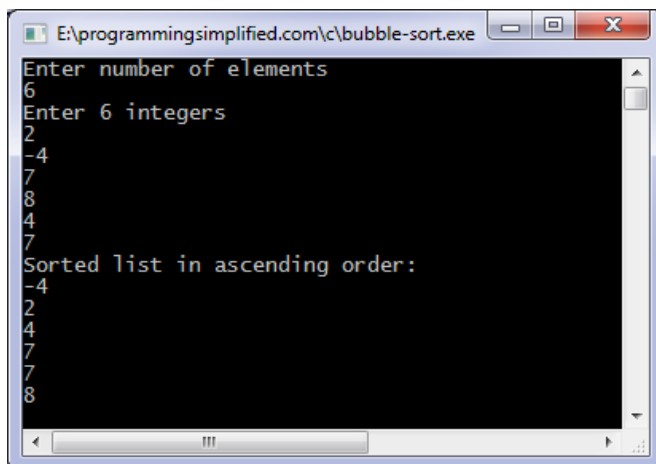
```
    printf("Enter number of elements\n");
```

```
    scanf("%d", &n);
```

```

printf("Enter %d integers\n", n);
for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
for (c = 0 ; c < n - 1; c++)
{
    for (d = 0 ; d < n - c - 1; d++)
    {
        if (array[d] > array[d+1]) /* For decreasing order use '<' instead of '>' */
        {
            swap    = array[d];
            array[d] = array[d+1];
            array[d+1] = swap;
        }
    }
}
printf("Sorted list in ascending order:\n");
for (c = 0; c < n; c++)
    printf("%d\n", array[c]);
return 0;
}

```



```

E:\programmingsimplified.com\c\bubble-sort.exe
Enter number of elements
6
Enter 6 integers
2
-4
7
8
4
7
Sorted list in ascending order:
-4
2
4
7
7
8

```

2. ການຈັດລຽງລຳດັບຂໍ້ມູນແບບເລືອກ Selection sort

ການຈັດລຽງຂໍ້ມູນແບບ selection sort ແມ່ນຈະຄົ້ນຫາຂໍ້ມູນທີ່ຫລາຍທີ່ສຸດໃນຊຸດຂໍ້ມູນທີ່ຕ້ອງການຈັດລຽງ ແລ້ວນຳຂໍ້ມູນທີ່ຫລາຍທີ່ສຸດໄປໄວ້ໃນຕຳແໜ່ງຂວາສຸດໃນຊຸດຂໍ້ມູນທີ່ຕ້ອງການຈັດລຽງ (ໃນກໍລະນີຫນ້ອຍໄປຫາຫລາຍ) ເຊິ່ງຂໍ້ມູນທີ່ຫລາຍທີ່ສຸດຈະສະຫລັບຕຳແໜ່ງກັບຊຸດຂໍ້ມູນໂຕສຸດທ້າຍ, ຂັ້ນຕອນຖັດໄປຫລັງຈາກຈັດລຽງຂໍ້ມູນຫລາຍທີ່ສຸດແລ້ວ ແມ່ນຄົ້ນຫາຂໍ້ມູນທີ່ຫລາຍທີ່ສຸດໃນລຳດັບຕໍ່ໄປໂດຍຍົກເວັ້ນຕຳແໜ່ງທີ່ຈັດແລ້ວກ່ອນຫນ້ານີ້ ເຮັດແບບນີ້ຈົນກະທັ່ງຈົບຄົບທຸກຂໍ້ມູນໃນຊຸດນັ້ນ.

ຕົວຢ່າງ: ການລຽງຂໍ້ມູນຈາກຫນ້ອຍໄປຫາຫລາຍ ກຳນົດເລກ 5 ໂຕ ຄື: 5 4 3 2 1

Selection sort ເທື່ອທີ 1

1 4 3 2 5

Selection sort ເທື່ອທີ 2

1 2 3 4 5

ຜົນຈາກການຈັດລຽງ **Selection sort** ໄດ້: 1 2 3 4 5

ໂປແກຣມ Selection sort ໃນພາສາ c

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

```
int main()
```

```
{
```

```
    int array[100], n, c, d, position, t;
```

```
    printf("Enter number of elements\n");
```

```
    scanf("%d", &n);
```

```
    printf("Enter %d integers\n", n);
```

```
    for (c = 0; c < n; c++)
```

```
        scanf("%d", &array[c]);
```

```
    for (c = 0; c < (n - 1); c++) // finding minimum element (n-1) times
```

```
    {
```

```
        position = c;
```

```
        for (d = c + 1; d < n; d++)
```

```
        {
```

```
            if (array[position] > array[d])
```

```

        position = d;
    }
    if (position != c)
    {
        t = array[c];
        array[c] = array[position];
        array[position] = t;
    }
}

printf("Sorted list in ascending order:\n");
for (c = 0; c < n; c++)
    printf("%d\n", array[c]);
return 0;
}

```

```

E:\programmingsimplified.com\c\selection-sort.exe
Enter number of elements
10
Enter 10 integers
12
8
-6
2
4
5
3
7
4
2
Sorted list in ascending order:
-6
2
2
3
4
4
5
7
8
12

```

3. ການຈັດລຽງລຳດັບຂໍ້ມູນແບບເຝິມເຂົ້າ Insertion sort

ການຈັດລຽງຂໍ້ມູນແບບ Insertion sort ຈະເຮັດວຽກໂດຍການແບ່ງຂໍ້ມູນໃນຊຸດຂໍ້ມູນອອກເປັນສອງພາກສ່ວນຄື: ສ່ວນທີ່ຈັດລຽງແລ້ວ ແລະ ສ່ວນນີ້ຍັງບໍ່ທັນຈັດລຽງ, ແນ່ນອນໃນຕອນແລກສ່ວນທີ່ຈັດລຽງຈະໜ່ວຍກວ່າສ່ວນທີ່ບໍ່ທັນຈັດລຽງເພາະມີພຽງໂຕດຽວ ແລ້ວ ຈະເລີ່ມຈັບເອົາຂໍ້ມູນໂຕ 1 ທີ່ບໍ່ທັນຈັດ

ລຽງມາປຽບທຽບເພື່ອຫາຕຳແໜ່ງທີ່ເໝາະສົມກັບການເພີ່ມເຂົ້າຫຼັງຈາກເພີ່ມຕາມຕຳແໜ່ງທີ່ເໝາະສົມ
ແລ້ວຂຶ້ນຕອນຕໍ່ໄປກໍ່ຈະຈັບເອົາຂໍ້ມູນທີ່ບໍ່ທັນລຽງມາປະຕິບັດອີກຄັ້ງເຮັດຕໍ່ໄປເລື້ອຍໆຈົນຈົບ.

ຕົວຢ່າງ: ການລຽງຂໍ້ມູນຈາກຫນ່ວຍໄປຫາຫລາຍ ກຳນົດເລກ 5 ໂຕ ຄື: **5 4 3 2 1**

Insertion sort ເທື່ອທີ 1

(5)4321 > (54)321 > (45)321

Insertion sort ເທື່ອທີ 2

(45)321 > (453)21 > (345)21

Insertion sort ເທື່ອທີ 3

(345)21 > (3452)1 > (2345)1

Insertion sort ເທື່ອທີ 4

(2345)1 > (23451) > (12345)

ຜົນຈາກການຈັດລຽງ **Insertion sort** ໄດ້: **1 2 3 4 5**

ໂປແກຣມ Insertion sort ໃນພາສາ c

```
#include <stdio.h>

int main()
{
    int n, array[1000], c, d, t, flag = 0;
    printf("Enter number of elements\n");
    scanf("%d", &n);
    printf("Enter %d integers\n", n);
    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);
    for (c = 1 ; c <= n - 1; c++) {
        t = array[c];
        for (d = c - 1 ; d >= 0; d--) {
            if (array[d] > t) {
                array[d+1] = array[d];
```

```

        flag = 1;
    }
    else
        break;
}
if (flag)
    array[d+1] = t;
}
printf("Sorted list in ascending order:\n");
for (c = 0; c <= n - 1; c++) {
    printf("%d\n", array[c]);
}
return 0;
}

```

```

E:\programmingsimplified.com\c\insertion-sort.exe
Enter number of elements
5
Enter 5 integers
4
3
-1
2
1
Sorted list in ascending order:
-1
1
2
3
4

```

4. ການຈັດລຽງລຳດັບຂໍ້ມູນແບບ chell sort

ການຈັດລຽງແບບ shell sort ແມ່ນໄດ້ຮັບການປັບປຸງມາຈາກ Insertion sort. ຂັ້ນຕອນການຈັດລຽງຂໍ້ມູນແບບ shell sort ແມ່ນຈະເຮັດການຫາຄ່າຂໍ້ມູນ gap ໂຕທຳອິດໂດຍເຮົາຈະໃຊ້ຄ່າເຄິ່ງ 1 ຂອງ gap ສົມມຸດວ່າຂໍ້ມູນມີ 10 ໂຕ ເຄິ່ງ 1 ຂອງຂໍ້ມູນແມ່ນ 5 ເຊິ່ງສາມາດໃຊ້ສຸດ $gap = n/2$ (n ແມ່ນຊຸດຂອງຂໍ້ມູນ) ດັ່ງນັ້ນ gap ໂຕທຳອິດແມ່ນ $10/2=5$ ຈາກນັ້ນຈັດລຽງຂໍ້ມູນ gap ຊຸດທຳອິດໃຫ້ສຳເລັດ, ຈາກນັ້ນເຮັດການຫາຄ່າຂອງ gap ໃຫ້ມ gap ໂຕທີ 2 ຈະມີຄ່າເທົ່າກັບເຄິ່ງ 1 ຂອງ gap ໂຕທີສອງ (ກໍລະນີຫານສອງແລ້ວມີເສດໃຫ້ປັດເສດຖິ້ມ).

ຕົວຢ່າງ: ການລຽງຂໍ້ມູນຈາກຫນ້າໄປຫາຫລາຍ ກຳນົດເລກ 10 ໂຕ ຄື: 10 9 8 7 6 5 4 3 2 1

shell sort ເທື່ອທີ 1

gap=5 ແມ່ນຈະໄດ້ 5 ແຖວ

10 5 5 10

9 4 4 9

8 3 -> 3 8

7 2 2 7

6 1 1 6

ໄດ້: 5 4 3 2 1 10 9 8 7 6

shell sort ເທື່ອທີ 2

Gap2=2.5 ແມ່ນຈະໄດ້ 2 ແຖວ

5 3 1 9 7 -> 1 3 5 7 9

4 2 10 8 6 2 4 6 8 10

ໄດ້: 1 2 3 4 5 6 7 8 9 10

ໂປແກຣມ shell sort ໃນພາສາ c

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

```
void shellsort(int arr[], int num)
```

```
{
```

```
    int i, j, k, tmp;
```

```
    for (i = num / 2; i > 0; i = i / 2)
```

```
    {
```

```
        for (j = i; j < num; j++)
```

```
        {
```

```
            for(k = j - i; k >= 0; k = k - i)
```

```
            {
```

```
                if (arr[k+i] >= arr[k])
```

```
                    break;
```



```

        else
        {
            tmp = arr[k];
            arr[k] = arr[k+i];
            arr[k+i] = tmp;
        }
    }
}

int main()
{
    int arr[30];
    int k, num;
    printf("Enter total no. of elements : ");
    scanf("%d", &num);
    printf("\nEnter %d numbers: ", num);
    for (k = 0 ; k < num; k++)
    {
        scanf("%d", &arr[k]);
    }
    shellsort(arr, num);
    printf("\n Sorted array is: ");
    for (k = 0; k < num; k++)
        printf("%d ", arr[k]);
    return 0;
}

```

```
D:\file studen2_2\Algorithm2_2\Project2.exe
Enter total no. of elements : 10
Enter 10 numbers: 10 9 8 7 6 5 4 3 2 1
Sorted array is: 1 2 3 4 5 6 7 8 9 10
-----
Process exited after 18.14 seconds with return value 0
Press any key to continue . . .
```

5. ການຈັດລຽງລຳດັບຂໍ້ມູນແບບQuick sort

ການຈັດລຽງຂໍ້ມູນແບບ Quick sort ເປັນການຈັດລຽງທີ່ໃຊ້ເວລາໜ້ອຍເໝາະສຳລັບຂໍ້ມູນທີ່ມີຈຳນວນຫລາຍຕ້ອງການຄວາມໄວໃນການເຮັດວຽກ; ການເຮັດວຽກວິທີນີ້ແມ່ນຈະເລືອກຂໍ້ມູນທຳອິດໃນຊຸດຂໍ້ມູນມາເປັນ Pivot ແລ້ວເຮັດການຈັກລຽງຫາຕຳແໜ່ງທີ່ຖືກຕ້ອງໃຫ້ກັບຂໍ້ມູນ; ເຊິ່ງການຈັດລຽງນັ້ນ ຈະເຮັດໂດຍການຄົ້ນຫາຂໍ້ມູນທີ່ນ້ອຍກວ່າຈາກທາງເບື້ອງຂວາແລ້ວສະສະຫລັບຕຳແໜ່ງ ຈາກນັ້ນກໍຈະຄົ້ນຫາຂໍ້ມູນທີ່ໃຫຍ່ກວ່າຈາກທາງເບື້ອງຊາຍເຫັນແລ້ວກໍຈະສະສະຫລັບຕຳແໜ່ງ ເຮັດໄປເລື້ອຍໆ ຈາກນັ້ນຈະໄດ້ຂໍ້ມູນສອງສ່ວນ ຄື: ສ່ວນແລກແມ່ນນ້ອຍກວ່າ pivot ແລະ ສ່ວນທີ່ສອງແມ່ນໃຫຍ່ກ່ອນ pivot ຫລັງຈາກນັ້ນນຳແຕ່ລະສ່ວນຢ່ອຍໄປແບ່ງຢ່ອຍໃນລັກສະນະດຽວກັນໄປເລື້ອຍໆຈົນຈັດລຽງໄດ້.

ຕົວຢ່າງ: ການລຽງຂໍ້ມູນຈາກໜ້ອຍໄປຫາຫລາຍ ກຳນົດເລກ 8 ໂຕ ຄື: **6 3 9 5 8 2 4 7**

ກຳນົດ 6 ເປັນ Pivot

6 3 9 5 8 2 4 7

ປຽບທຽບຂໍ້ມູນ **4 3 9 5 8 2 6 7**

ປຽບທຽບຂໍ້ມູນ **4 3 6 5 8 2 9 7**

ປຽບທຽບຂໍ້ມູນ **4 3 2 5 8 6 9 7**

ປຽບທຽບຂໍ້ມູນ **4 3 2 5 6 8 9 7**

ໄດ້ **4 3 2 5 6 8 9 7**

ຈະໄດ້ **4 3 2 5** ສ່ວນທີ 1 ນ້ອຍກວ່າ Pivot **6**

ຈະໄດ້ 8 9 7 ສ່ວນທີ 2 ໃຫຍ່ກວ່າ Pivot 6

ນຳສ່ວນທີ 1 ມາປຽບທຽບຂໍ້ມູນ 4 3 2 5

ກຳນົດ 4 ເປັນ Pivot

4 3 2 5

ປຽບທຽບຂໍ້ມູນ 2 3 4 5

ໄດ້ 2 3 4 5

ນຳສ່ວນທີ 2 ມາປຽບທຽບຂໍ້ມູນ 8 9 7

ກຳນົດ 8 ເປັນ Pivot

8 9 7

ປຽບທຽບຂໍ້ມູນ 7 9 8

ປຽບທຽບຂໍ້ມູນ 7 8 9

ໄດ້ 7 8 9

ນຳແຕ່ລະສ່ວນມາລວມກັນຈະໄດ້ 2 3 4 5 6 7 8 9

ໂປແກຣມ Quick sort ໃນພາສາ c

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

```
void quicksort (int [], int, int);
```

```
int main()
```

```
{
```

```
    int list[50];
```

```
    int size, i;
```

```
    printf("Enter the number of elements: ");
```

```
    scanf("%d", &size);
```

```
    printf("Enter the elements to be sorted:\n");
```

```
    for (i = 0; i < size; i++)
```

```
    {
```

```

        scanf("%d", &list[i]);
    }
    quicksort(list, 0, size - 1);
    printf("After applying quick sort\n");
    for (i = 0; i < size; i++)
    {
        printf("%d ", list[i]);
    }
    printf("\n");
    return 0;
}

void quicksort(int list[], int low, int high)
{
    int pivot, i, j, temp;
    if (low < high)
    {
        pivot = low;
        i = low;
        j = high;
        while (i < j)
        {
            while (list[i] <= list[pivot] && i <= high)
            {
                i++;
            }
            while (list[j] > list[pivot] && j >= low)
            {
                j--;
            }

```

```

    }
    if (i < j)
    {
        temp = list[i];
        list[i] = list[j];
        list[j] = temp;
    }
}
temp = list[j];
list[j] = list[pivot];
list[pivot] = temp;
quicksort(list, low, j - 1);
quicksort(list, j + 1, high);
}
}

```

```

D:\file studen2_2\Algorithm2_2\QuickSort.exe
Enter the elements to be sorted:
6 3 9 5 8 2 4 7
After applying quick sort
2 3 4 5 6 7 8 9

-----
Process exited after 30.12 seconds with return value 0
Press any key to continue . . .

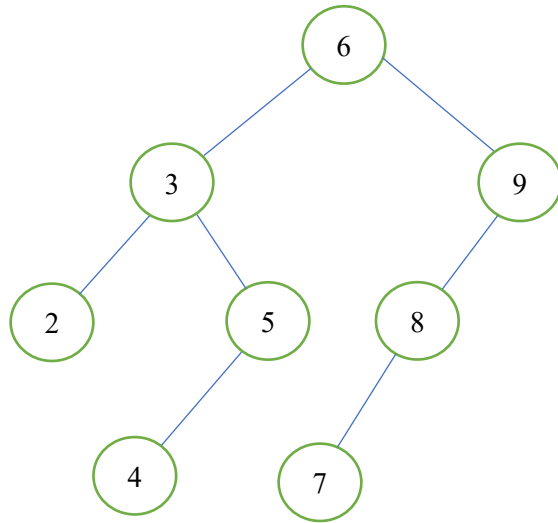
```

6. ການຈັດລຽງລຳດັບຂໍ້ມູນແບບ heap sort

ການລຽງລຳດັບແບບ heap sort ເປັນວິທີການລຽງທີ່ໃຊ້ປະໂຫຍດຈາກໂຄງສ້າງຕົ້ນໄມ້ແບບ binary search tree ໂດຍຈະນຳຂໍ້ມູນຈາກຊຸດຂໍ້ມູນມາສ້າງເປັນ binary search tree ແລ້ວໃຊ້ການເຂົ້າເຖິງຂໍ້ມູນໂດຍໃຊ້ສູດ L N R ເພື່ອຈັດລຽງຂໍ້ມູນ.

ຕົວຢ່າງ: ການລຽງຂໍ້ມູນຈາກຫນ່ວຍໄປຫາຫລາຍ ກຳນົດເລກ 8 ໂຕ ຄື: **6 3 9 5 8 2 4 7**

ນຳຊຸດຂໍ້ມູນມາສ້າງເປັນ binary search tree ຈະໄດ້



ຫລັງຈາກນັ້ນຈະໃຊ້ວິທີການເຂົ້າເຖິງຂໍ້ມູນແບບ In-Order(L N R) ເພື່ອຈັດລຽງຂໍ້ມູນ

ຈະໄດ້ 2 3 4 5 6 7 8 9

ໂປແກຣມ heap sortໃນພາສາ c

```
#include<stdio.h>
void create(int []);
void down_adjust(int [],int);
int main()
{
    int heap[30],n,i,last,temp;
    printf("Enter no. of elements:");
    scanf("%d",&n);
    printf("\nEnter elements:");
    for(i=1;i<=n;i++)
        scanf("%d",&heap[i]);
    //create a heap
    heap[0]=n;
    create(heap);
    //sorting
    while(heap[0] > 1)
    {
        //swap heap[1] and heap[last]
        last=heap[0];
        temp=heap[1];
        heap[1]=heap[last];
        heap[last]=temp;
        heap[0]--;
```

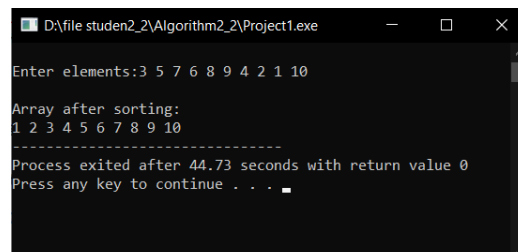
```

        down_adjust(heap,1);
    }
    //print sorted data
    printf("\nArray after sorting:\n");
    for(i=1;i<=n;i++)
        printf("%d ",heap[i]);
    return 0;
}
void create(int heap[])
{
    int i,n;
    n=heap[0]; //no. of elements

    for(i=n/2;i>=1;i--)
        down_adjust(heap,i);
}
void down_adjust(int heap[],int i)
{
    int j,temp,n,flag=1;
    n=heap[0];

    while(2*i<=n && flag==1)
    {
        j=2*i; //j points to left child
        if(j+1<=n && heap[j+1] > heap[j])
            j=j+1;
        if(heap[i] > heap[j])
            flag=0;
        else
        {
            temp=heap[i];
            heap[i]=heap[j];
            heap[j]=temp;
            i=j;
        }
    }
}

```



```

D:\file studen2_2\Algorithm2_2\Project1.exe
Enter elements:3 5 7 6 8 9 4 2 1 10

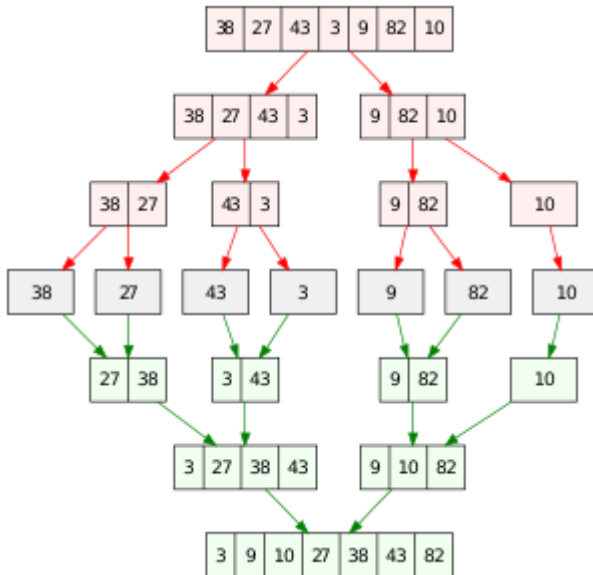
Array after sorting:
1 2 3 4 5 6 7 8 9 10
-----
Process exited after 44.73 seconds with return value 0
Press any key to continue . . .

```

7. ການຈັດລຽງລຳດັບຂໍ້ມູນແບບ merge sort

ການຈັດລຽງຂໍ້ມູນແບບປະສານ ຫລື merge sort ຄືການຈັດລຽງຂໍ້ມູນທີ່ນຳເອົາຊຸດຂໍ້ມູນນັ້ນໆມາແບ່ງຂໍ້ມູນອອກເປັນສອງສ່ວນກ່ອນ ເຊິ່ງແຕ່ລະສ່ວນກໍ່ຈະແບ່ງຍ່ອຍຂໍ້ມູນອອກເປັນ 2 ສ່ວນເລື້ອຍໆຈົນກະທັ້ງບໍ່ສາມາດແບ່ງໄດ້ອີກແລ້ວຄ່ອຍຈັດລຽງຂໍ້ມູນໃນສ່ວນຍ່ອຍຈາກນັ້ນນຳເອົາຂໍ້ມູນສ່ວນຍ່ອຍທີ່ລຽງໄວ້ແລ້ວມາລວມເຂົ້າກັນ ເຊິ່ງ ຈະລຽງພ້ອມໄປສານຂໍ້ມູນໄປພ້ອມພ້ອມກັນຈົນກະທັ້ງຂໍ້ມູນລວມເປັນຊຸດດຽວ.

ຕົວຢ່າງ: ການລຽງຂໍ້ມູນຈາກຫນ່ວຍໄປຫາຫລາຍ ກຳນົດເລກ 7 ໂຕ ຄື: 38 27 43 3 9 82 10



ໂປແກຣມ merge sort ໃນພາສາ c

```
#include<stdio.h>
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
int main()
{
    int a[30],n,i;
    printf("Enter no of elements:");
    scanf("%d",&n);
    printf("Enter array elements:");

    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    mergesort(a,0,n-1);

    printf("\nSorted array is :");
    for(i=0;i<n;i++)
        printf("%d ",a[i]);
}
```



```

        return 0;
    }

void mergesort(int a[],int i,int j)
{
    int mid;

    if(i<j)
    {
        mid=(i+j)/2;
        mergesort(a,i,mid);           //left recursion
        mergesort(a,mid+1,j); //right recursion
        merge(a,i,mid,mid+1,j);      //merging of two sorted sub-arrays
    }
}

void merge(int a[],int i1,int j1,int i2,int j2)
{
    int temp[50]; //array used for merging
    int i,j,k;
    i=i1; //beginning of the first list
    j=i2; //beginning of the second list
    k=0;
    while(i<=j1 && j<=j2) //while elements in both lists
    {
        if(a[i]<a[j])
            temp[k++]=a[i++];
        else
            temp[k++]=a[j++];
    }
    while(i<=j1) //copy remaining elements of the first list
        temp[k++]=a[i++];
    while(j<=j2) //copy remaining elements of the second list
        temp[k++]=a[j++];

    //Transfer elements from temp[] back to a[]
    for(i=i1,j=0;i<=j2;i++,j++)
        a[i]=temp[j];
}

```

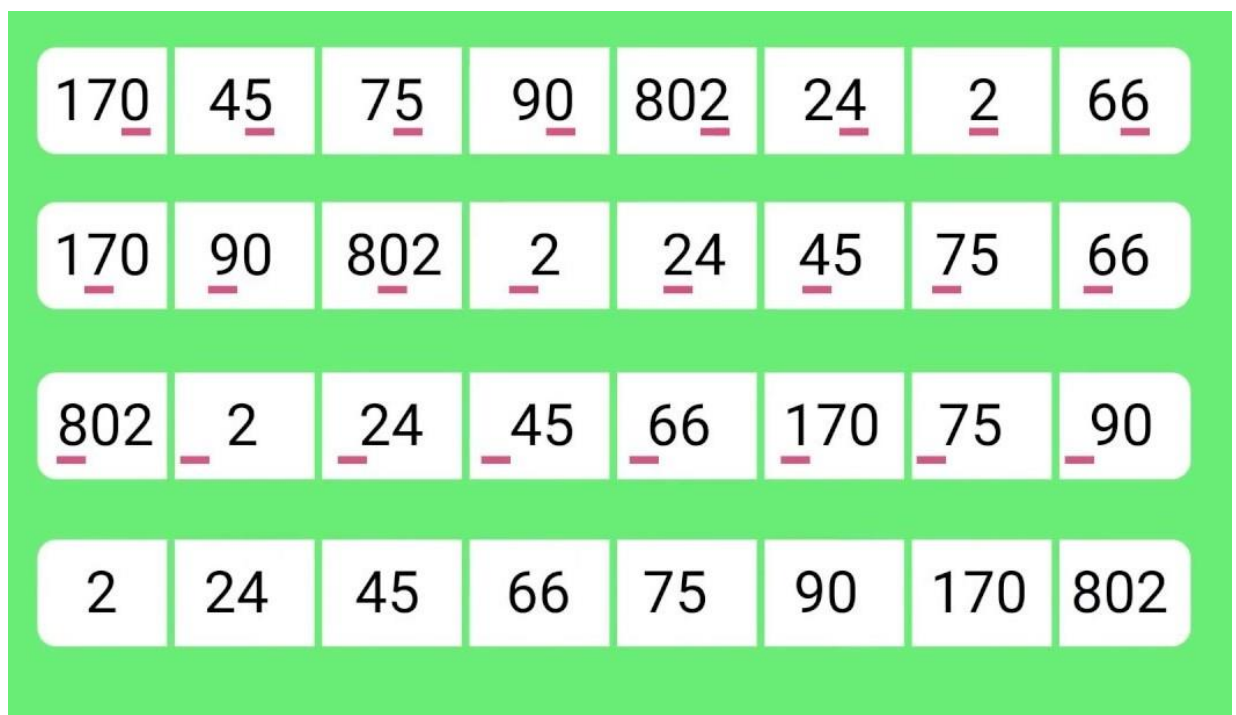
```
D:\file studen2_2\Algorithm2_2\Project1.exe
Enter no of elements:5
Enter array elements:6 4 9 2 7

Sorted array is :2 4 6 7 9
-----
Process exited after 14.73 seconds with return value 0
Press any key to continue . . .
```

8. ການຈັດລຽງລຳດັບຂໍ້ມູນແບບ Radix Sort

ເປັນການຈັດລຽງຂໍ້ມູນທີ່ຝຳລະນາຂໍ້ມູນເທື່ອລະຫລັກ ເລີ່ມຝຳລະນາຈາກຫລັກທີ່ນ້ອຍທີ່ສຸດກ່ອນ ໂດຍເລີ່ມຈາກຫລັກໜ່ວຍກ່ອນ ການຈັດລຽງຈະນຳຂໍ້ມູນເຂົ້າມາເທື່ອລະໂຕ ເກັບໄວ້ຕາມແຕ່ລະກຸ່ມໂດຍເລີ່ມຈາກກຸ່ມທີ່ມີຄ່ານ້ອຍສຸດກ່ອນແລ້ວລຽງໄປເລື້ອຍໆຈົນຫມົດທຸກກຸ່ມ, ຫລັງຈາກນັ້ນນຳຂໍ້ມູນແຕ່ລະກຸ່ມມາໂຮມເຂົ້າກັນເປັນຊຸດມູນ ແລ້ວເຮັດຫລັກຕໍ່ໄປຈົນຈົບ (ສິບ, ຮ້ອຍ...).

ຕົວຢ່າງ: ການລຽງຂໍ້ມູນຈາກຫນ້ອຍໄປຫາຫລາຍ ກຳນົດເລກ 8 ໂຕ ຄື: 170 45 75 90 802 24 2 66



ໂປແກຣມ Radix sortໃນພາສາ c

```
#include<stdio.h>
// Function to find largest element
int largest(int a[], int n)
{
    int large = a[0], i;
    for(i = 1; i < n; i++)
    {
        if(large < a[i])
            large = a[i];
    }
    return large;
}
// Function to perform sorting
void RadixSort(int a[], int n)
{
    int bucket[10][10], bucket_count[10];
    int i, j, k, remainder, NOP=0, divisor=1, large, pass;
    large = largest(a, n);
    printf("The large element %d\n", large);
    while(large > 0)
    {
        NOP++;
        large/=10;
    }
    for(pass = 0; pass < NOP; pass++)
    {
        for(i = 0; i < 10; i++)
        {
            bucket_count[i] = 0;
        }
        for(i = 0; i < n; i++)
        {
            remainder = (a[i] / divisor) % 10;
            bucket[remainder][bucket_count[remainder]] = a[i];
            bucket_count[remainder] += 1;
        }
        i = 0;
        for(k = 0; k < 10; k++)
        {
            for(j = 0; j < bucket_count[k]; j++)
            {
                a[i] = bucket[k][j];
                i++;
            }
        }
        divisor *= 10;
    }
}
```

```
        for(i = 0; i < n; i++)
            printf("%d ",a[i]);
        printf("\n");
    }
}
//program starts here
int main()
{
    int i, n, a[10];
    printf("Enter the number of elements :: ");
    scanf("%d",&n);
    printf("Enter the elements :: ");
    for(i = 0; i < n; i++)
    {
        scanf("%d",&a[i]);
    }
    RadixSort(a,n);
    printf("The sorted elements are :: ");
    for(i = 0; i < n; i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```