ADA 30. september 2024

Sortering

Introducerende antagelse

Sortering i det virkelig liv finder sted på alle mulige forskellige datatyper.

For at undgå begrebsforvirring arbejder vi udelukkende med sortering af heltal (integers), under den antagelse, at alle de beskrevne principper kan anvendes for alle andre datatyper.

Inversion

Lad os betragte denne liste af heltal:

34, 8, 64, 51, 32, 21

Tallene er helt klar usorterede, så lad os prøve at se, om vi kan finde et mål for, hvor usorterede de er.

Til at hjælpe os definerer vi et begreb kaldet *inversion*. En inversion er et ordnet talpar (i,j), hvorom det gælder, at i < j, men a[i] > a[j].

I vores eksempel forekommer der følgende inversions:

(34,8) (34,32) (34,21) (64,51) (64,32) (64,21) (51,32) (51,21) (32,21)

Sortering

- Jo flere inversions listen indeholder, desto mere usorteret er den.
- Sorteringsdisciplinen går ud på at reducere antallet af inversions i en liste til nul.

Mere om inversions

- Hvad er antallet af inversions i denne liste: 78,63,61,45,25,11?
- Det gennemsnitlige antal inversions i en liste bestående af N forskellige heltal er N(N-1)/4.
- Hvis én operation kan eliminere én inversion, så må den slags sortering have kvadratisk tidskompleksitet $O(N^2)$.
- Hvis mere end én inversion kan elimineres af én operation, kan vi opnå bedre resultater end kvadratisk tidskompleksitet.

$O(N^2)$

Insertionsort

Selectionsort

Bubblesort

Bubblesort

```
1   public void bubbleSort(int array[]) // Java
2   {
3           for (int i = 0; i < array.length - 1; i++)
4           {
5                   boolean swapped = false;
6                   for (int j = 0; j < array.length – i - 1; j++)
7                   {
8                           if (array[j] > array[j+1])
9                           {
10                                  swap(array,j,j+1);
11                                  swapped = true;
12                          }
13                  }
14                  if (!swapped)
15                          return;
16          }
17  }
```

O(N logN)

Mergesort

Heapsort

Quicksort

Mergesort

```
private static <AnyType extends Comparable<? super AnyType>>
void mergeSort( AnyType [ ] a, AnyType [ ] tmpArray, int left, int right )
{
    if( left < right )
    {
        int center = ( left + right ) / 2;
        mergeSort( a, tmpArray, left, center );
        mergeSort( a, tmpArray, center + 1, right );
        merge( a, tmpArray, left, center + 1, right );
    }
}

/**
 * Mergesort algorithm.
 * @param a an array of Comparable items.
 */
public static <AnyType extends Comparable<? super AnyType>>
void mergeSort( AnyType [ ] a )
{
    AnyType [ ] tmpArray = (AnyType[]) new Comparable[ a.length ];

    mergeSort( a, tmpArray, 0, a.length - 1 );
}
```

Merge

```
// Precondition: no duplicates
public static int[] merge (int[] a, int[] b)  //Java
{
    int [] m = new int[a.length+b.length];
    int pointerA = 0; int pointerB = 0; int pointerM = 0;

    while (pointerA < a. length && pointerB < b.length)
        if (a[pointerA] < b[pointerB])
            m[pointerM++] = a[pointerA++];
        else
            m[pointerM++] = b[pointerB++];

    while (pointerA < a.length)
        m[pointerM++] = a[pointerA++];

    while (pointerB < b.length)
        m[pointerM++] = b[pointerB++];

    return m;
}
```

Heapsort

Pladsproblemet kan løses ved at genbruge tabellen fra enden af.

Analysen er besværlig; men performance er relativt konstant.

Det gennemsnitlige antal sammenligninger er (muligvis ☺):

$$2N \log N - O(N \log \log N)$$

Mergesort

Er den sortering med de færreste sammenligninger og anvendes af Javas standardbibliotek.

C++ bruger QuickSort.

Analyse: best case, average case, worst case.

Quicksort

```
1   public void quickSort(int array[], int left, int right)
2   {
3           if (left + CUTOFF < right)
4           {
5                   int pivot = median3(array, left, right);
6                   int i = left, j = right -1;
7                   for ( ; ; )
8                   {
9                           while( array[i] < pivot) i++;
10                          while( array[j] > pivot) j--;
11                          If (i < j)
12                                  swap(array, i, j);
13                          else
14                                  break;
15                  }
16                  swap(array, i, right - 1);
17
18                  quickSort(array, left, i - 1);
19                  quickSort(array, i + 1, right);
20          }
21          else
22                  insertionSort(array, left, right);
23  }
```

Shellsort

Shellsort er en ‘multi-pass’ algoritme. Hvert pass er en insertion sort af sekvenser bestående af hvert *h.* element for et for et fixed fast mellemrum *h* (også kaldt inkrementet). Dette kaldes *h*-sortering.

Shellsort

Et eksempel på Shellsort med mellemrum 5, 3 og 1 er vist nedenfor.

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12
input	62	83	18	53	07	17	95	86	47	69	25	28
efter 5 sort	17	28	18	47	07	35	83	86	53	69	62	95
efter 3 sort	17	07	18	47	28	25	69	62	53	83	86	95
efter 1 sort	07	17	18	25	28	47	53	62	29	83	86	95

The first pass, 5-sorting, performs insertion sort on separate subarrays (a_1, a_6, a_{11}), (a_2, a_7, a_{12}), (a_3, a_8), (a_4, a_9), (a_5, a_{10}). For instance, it changes the subarray (a_1, a_6, a_{11}) from (62, 17, 25) to (17, 25, 62). The next pass, 3-sorting, performs insertion sort on the subarrays (a_1, a_4, a_7, a_{10}), (a_2, a_5, a_8, a_{11}), (a_3, a_6, a_9, a_{12}). The last pass, 1-sorting, is an ordinary insertion sort of the entire array (a_1,..., a_{12}).

Find two errors.

Shellsort

- Som eksemplet illustrerer, er de subarrays, som Shellsort opererer på små til at starte med; senere bliver de større, men næsten sorterede. I begge til fælde er insertion sort effektiv.
- Shellsort is unstabil: det kan forekommer, at den ændrer rækkefølgen af elementer med samme værdi. Den har en 'naturlig' adfærd (lige som bubblesort), eftersom den afvikles hurtigere, hvis tabellen er relativt sorteret (få inversions).

bucketSort()

- Under særlige omstændigheder kan vi gøre det bedre end O(N log N).
- Hvis vi skal sortere positive heltal med en maksimumsværdi M, kan vi løse problemet ved først at lave et array med plads til M elementer.
- Tallene læses fra ende til anden, og når værdien X læses, så inkrementeres M[X] med en.
- Når alle tal er læst, kan det sorterede array etableres i overensstemmelse med antallet af forekomster i indekserne.
- Tidskomplekstet O(M+N) -> O(N).

Sortering

- Den nedre grænse for sortering, hvis der kun anvendes sammenligninger, er: $\Omega(N \log N)$.
- Det kan vises ved at tilføje alle sammenligninger til et binært træ (sektion 7.8).
- Der er forskel på *lower bound* and *best case*.

Konklusion

- Inversions
- $O(N^2)$
- O(N log N)
- O(N)
- Vælg den mest hensigtsmæssige på baggrund af det talmateriale, der skal sorteres.
- Problemet er næppe vigtigt, hvis der er færre end 100.000 elementer, og sorteringen derfor kan udføres i memory.

Animated Hungarian QuickSort

https://www.youtube.com/watch?v=ywWBy6J5gz8

Exercises

1. En tabel indeholder N sorterede heltal. Opgaven går ud på at afgøre om tabellen indeholder to tal, hvis sum er lig med parameteren X. Metoden skal skrives i to udgaver: én med kvadratisk tidskompleksitet og en med lineær.
2. Ordene *stale og least* er anagrammer, dvs. de indeholder nøjagtig de samme karakterer. Skriv en metode, som kan afgøre, om to ord (strings) er anagrammer.

Øvelser fra lærebogen med stigende sværhedsgrad.

7.44

7.38

7.43*

7.40*