Sorting Algorithms(Insertion sort, shell sort, merge sort)

Renu Jain, Data Structures and algorithms, JKLU, Jaipur

Insertion sort

- Which sorting algorithm will you use to sort a stack of 60 copies?
- Insertion sort assumes an empty array and keeps adding new data such that all the inserted data is sorted.
- It sorts a set of records by inserting records into an existing sorted file.
- A subarray of size 1 is considered sorted and more elements are added to this sorted file one by one keeping the array sorted.
- We start from the second element and insert it at the right place
- The process continues for all the elements from X[2] to X[n-1]
- After n-1 iterations, we get the sorted file

Insertion sort(example)

How do we put every element at the right place???

- We compare element ***a*** at location k, with element at location k-1, X[k-1]

- If (a >= X[k-1]) no change else shift X[k-1] to X[k] position and now a is compared with X[k-2] … and process continues….

25 67 12 34 7 98 45 80

Cont..

Step1: Examine the 2nd element, store it in a temporary variable ***a*** and compare it with the first element:

- **If(a >X[0])** no change
- ***else*** shift X[0] to location 1 and put ***a*** at X[0]

Step2: Examine the 3rd element, store it in a temporary variable ***a*** and compare it with the second element:

- ***If(a >X[1])*** no change
- e***lse*** shift X[1] to location 2 and compare ***a*** with X[0]

and continue…in the same way.

Insertion sort complexity

First step: Maximum one comparison

Second step: Maximum two comparisons

Kth step : Maximum k comparisons

Worst case: 1 +2 + 3 +… n-1

Best case: only one comparison in each step

Average case: between 1 and k comparisons in kth step

Note: Binary search can be used and linked list can be used to improve the sort.

Shell Sort

- Let us examine insertion sort more closely…
- **If we are trying to sort a very large file using insertion sort in which few elements are out of place, then…**
- *The number of comparisons and shifting movements will get reduced drastically*.
- Suppose a faculty has a class size of 1000 students and suppose there are 5 teaching assistants who are helping him/her in the correction. The faculty gives 200 unsorted copies to each TA and there may be two scenarios..

1. Every TA returns the copies unsorted
2. Every TA sorts the copies and then returns

Evaluate both the scenarios…

- It will be easier for the faculty to sort all the copies if each set is sorted
- We shall assume here that all the TAs combined their copies one after another and copies were given to each TA randomly
- Let us take an example:
- TA1: 2, 5, 8
- TA2: 3, 4, 9
- TA3: 1,6,7
- A faculty may make a pile as : 2,3,1,5,4,6,8,9,7 or 2,5,8,3,4,9,1,6,7 or some other permutation.

Cont…

- **Shell sort uses two concepts**:

1. If the size of file is not big (i.e.) n is small then there is not much difference between O(n^2) and O(n logn)
2. If the file is partially sorted, insertion sort is efficient.

Shell sort

- This method sorts separate subfiles of the original file
- These subfiles contain every kth element of the original file
- The value of k is called increment

Suppose we have a file of 23 elements and k =5, then we will have 5 files having elements as follows:

File1: x[0], x[5], x[10], x[15], x[20]

File2: x[1], x[6], x[11], x[16], x[21]

File3: x[2], x[7], x[12], x[17], x[22]

File4: x[3], x[8], x[13], x[18], x[23]

File5: x[4], x[9], x[14], x[19]

Shell sort

- Now File1 to File5 are sorted separately, we can use insertion sort or some other sort also
- A new smaller value of k is chosen and complete file is again partitioned into k-subfiles and each file is sorted separately again
- This process is repeated till k becomes 1 and then whole file is sorted using insertion sort
-

Example(shell sort)

- (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)
- 62 6 34 21 11 56 80 2 14 18 70 85 45 8 55 76 90
- K = 5
- File1: 62 56 70 76 :: 56 62 70 76
- File2: 6 80 85 90 :: 6 80 85 90
- File3: 34 2 45 :: 2 34 45
- File4: 21 14 8 :: 8 14 21
- File5: 11 18 55 :: 11 18 55
- New array after one iteration:

56 6 2 8 11 62 80 34 14 18 70 85 45 21 55 76 90

Cont…

- 56 6 2 8 11 62 80 34 14 18 70 85 45 21 55 76 90
- After taking K=3,
- File1: 56 8 80 18 45 76 :: 8 18 45 56 76 80
- File2: 6 11 34 70 21 90 :: 6 11 21 34 70 90
- File3: 2 62 14 85 55 :: 2 14 55 62 85

- New array after second iteration taking k=3

8 6 2 18 11 14 45 21 55 56 34 62 76 70 85 80 90

Cont…

- Now taking k=1,
- Applying insertion sort on the whole array:
- 2 6 8 11 14 18 21 34 45 55 56 62 70 76 80 85 90
- What are we gaining from Shell sort????

What are we gaining from Shell sort

- We know that simple insertion sort is highly efficient when applied on an almost sorted file
- For small n, there is not much difference between n^2 and nlogn
- In shell sort, when k is large individual files are very small, so insertion sort on these files is quite fast
- Each iteration of dividing into subfiles and sorting each, makes the entire file nearly sorted and due to this, insertion sort on final file gets efficient
- It is important to understand that if a file is partially sorted using an increment k and then subsequently sorted using an increment p, the file remains partially sorted on increment k, i.e. subsequent partial sorts do not disturb each other.
- Efficiency of shell sort is $n(\log n)^2$

Merge Sort

- Merging is the process of combining two or more sorted files into a third sorted file
- Let us see the process of merging two sorted files:
- (4, 8, 20, 40)   and (2, 9, 31, 35, 45)
- Algorithm:
- Assign a pointer to the start of each array as 0
- Have an empty array and initialize it's pointer to 0
- Compare the two elements—
- Copy the smaller one to third array and increment the pointer of that array and the target array
- Continue the process till the value of any pointer becomes equal to the size of its array
- Copy the rest of the array into target array

Merge Sort

- We can use this technique to sort a file in the following way:

1. Divide the file of size n, into 2 files of size n/2
2. Recursively divide each file till file of size 1
3. Merge the files of size 1 to get a sorted file of size 2, total n/2 files
4. Again combine adjacent files to create n/4 sorted files of size 4
5. Continue this … till we have a file of size n
6. This requires an auxiliary array of size n

Merge sort example (dividing)

40 30 12 45 56 34 78   1 85 98 17   10

[40, 30, 12, 45, 56, 34]   [78, 1, 85, 98,17,10]

[40, 30, 12]   [45, 56, 34]   [78, 1, 85] [98, 17, 10]

[40,30]   [12]   [45,56] [34]   [78,1] [85]   [98,17] [10]

[40] [30]   [12]   [45] [56]   [34]   [78] [1]   [85]   [98] [17]   [10]

Merging…

[40] [30]   [12]   [45] [56]   [34]   [78] [1]   [85]   [98] [17]   [10]

[30 40]   [12]   [45 56] [34]   [1 78]   [85]   [17 98] [10]

[12 30 40]   [34 45 56]   [1 78 85] [10 17 98]

[12 30 34 40 45 56]   [1 10 17 78 85 98]

[1 10 12 17 30 34 40 45 56 78 85 98]

Complexity of Merge sort

- In every pass of merging there are not more than ***n*** comparisons
- It obviously does not require more than $\boldsymbol{log_2 n}$ passes so overall complexity of merge sort is $\boldsymbol{n \log_2 n}$
- Merge sort requires approximately twice as many assignments as quicksort on average even if alternating merges go from array ***x*** to auxiliary array ***aux*** and from ***aux*** to ***x***
- Merge sort requires additional O(n) space whereas quicksort requires O(logn) space for stack.

Radix sort

- This sort is based on the values of actual digits in the positional representations of the numbers being sorted
- We perform the following actions on each digit of numbers beginning with the least significant digit and ending with most significant digit.
- We make ten queues one for each digit
- We take each number in the order in which it appears in the file and place it in one of the ten queues depending upon the value of digit currently being processed
- Then restore each queue to the original file starting with the queue of numbers with a digit 0 and ending with digit 9.
- When these actions are completed for each digit, the file is sorted

Example

- 25 57 48 37 12 7 920 86 303 4
- ***Queues based on least significant digit***

Q[0]: 920 Q[1]: Q[2]: 12 Q[3]: 303 Q[4]: 4

Q[5]: 25 Q[6]: 86 Q[7]: 7 37 57 Q[8]: 48 Q[9]:

- ***New array: 920 12 303 4 25 86 7 37 57 48***
- ***Queues based on next significant digit:***

Q[0]: 04 07 303 Q[1]: 12 Q[2]: 25 920 Q[3]: 37 Q[4]: 48

Q[5]: 57 Q[6]: Q[7]: Q[8]: 86 Q[9]:

- ***New array: 04 07 303 12 25 920 37 48 57 86***
- ***Queues based on last significant digit***:

Q[0]: 004 007 012 025 037 048 057 086 Q[1]: Q[2]: Q[3]: 303 Q[4]: Q[5]: Q[6]: Q[7]: Q[8]: Q[9]: 920

- ***Sorted array: 004 007 012 025 037 048 057 086 303 920***

Complexity of Radix sort

- The time requirement of Radix sort depends on the number of maximum digits **m** and number of elements **n** in the file
- The number of iterations are equal to number of maximum digits and in every iteration we process each element of array
- So, sort is approximately O(m*n)
- Hence, sort is efficient if number of digits are less
- m approximates log n so that O(m*n) comes out to be n logn

GATE 2020 questions

Consider the following function

```
int unknown (int n) {
int i, j, k = 0;
for (i = n / 2; i <= n; i + + )
for ( j = 2; j <= n; j = j * 2 )
k = k + n / 2;
return (k ) ;
}
```

- (A) Θ (n^2) (B) Θ (n^2 logn) (C) Θ (n^3) (D) Θ (n^3 logn)
-
-

- **Ans:(B)**

- If an array A contains the items 10, 4, 7, 23, 67, 12 and 5 in that order, what will be the resultant array A after third pass of insertion sort?
- A 67,12,10,5,4,7,23
- B 4,7,10,23,67,12,5
- C 4,5,7,67,10,12,23
- D 10,7,4,67,23,12,5