CST Part IA: Algorithm, SV 1
Joe Yan
2017-1-22

1 2010P1Q5

(a) Merge sort is based on merging two sorted sub-arrays to a big sorted array while leaving the two sub-array alone and forming the merged array in a temporary buffer.
By doing the previous step recursively, merge the array of size one to two, two to four etc and finally end up with a sorted array. (This is the bottom-top loop version. However the recursion or top-bottom version is done by keep dividing the input array to two sub-array with the same size and the merge them recursively)
Loop: 9 3 6 2 4 1 5 - 39 26 14 5 - 2369 145 - 1234569
Recursion: 9 3 6 2 4 1 5 - 9 3 6 2 4 1 5 - 9 36 2 4 1 5 - 369 2 4 1 5 - 369 24 1 5 - 369 24 15 - 369 2415 - 1234569

(b) Let the computation step for size n insertion sort to be f(n).

$$f(n) = f(n-1) + f(1) + O(n) = f(n-1) + O(n)$$

$$f(n) = \Sigma_{i=1}^{n} O(i) = O(\frac{n(n+1)}{2}) = O(n^2)$$

(c1) Let the computation step for size n merge sort to be f(n).$n = 2^m$

$$f(n) = 2f(n/2) + kn = 2f(2^{m-1}) + k \cdot 2^m$$

$$f(n) = 2^m + k \cdot m \cdot 2^m = n + kn \log n = O(n \log n)$$

The difference of linked list and array is that linked list can remove the node from any position from the data structure without leaving a hole in the memory but array cannot do so.
Suppose now we want to merge two linked list 24 13 to form one linked list. As 2 is larger than 1, so first unassign the node 1 from the linked list 13 and now we have 1 24 3. Keep going on 12 4 3 - 123 4 - 1234. As we are not creating any duplication of node, The only auxiliary space is needed for references or pointers signed to the start of the linked list. Overall the space complexity is O(n).

(c2) There are several aspects affecting whether we should do so.
The first is the size of the data input. The convert from array to linked list should take O(n) time complexity. If the data input is small, it is probably not worth doing convert as the time cost of converting may be close to or even more than the time cost of sorting.
The second is the size per individual data compare with the size of two pointers or references. (assigned to the previous and next node) If the data is byte (8 bits) however the size of a pointer under 64-bit OS is 8 bits under such assumption the converting triple the size of the data and is actually a waste. If the data is objects occupying large memory then it is probably worth doing the convert.

2 2008P11Q7

(a) Function partition takes a array and picks a pivot p (can be any element in the array) and rearrange the array to element smaller or equal to p concat p concat element larger than p. After the partition, the array has exactly the same elements with different order. The partition should return the position of the pivot.

(b)
```
quicksort(arr[],beg,end){
        if(end-beg < 1) return

        p = partition(arr[],beg,end); //position of the pivot
        quicksort(arr[],beg,p-1);
        quicksort(arr[],p+1,end);
}
```

Notice $p \in [beg, end] \Rightarrow [beg, p-1] \subset [beg, end] \wedge [p+1, end] \subset [beg, end]$
As the size of array each quicksort processing is always decreasing, finally the last many quicksort(s) will take either an empty array or with one element. Now the quicksort will return and so terminated.

(c) The recursion time complexity function under the best case (Every partition will finally put the pivot in the middle of the array):

$$f(n) = 2f(n/2) + kn$$

So the worst case should be the pivot for every partition end up to the begin or end of the array.

$$f(n) = f(1) + f(n-1) + kn = nf(1) + \frac{kn(n+1)}{2} = O(n^2)$$

To improve the behaviour, take several samples from random positions and pick the median of them as the pivot will significantly reduce the probability of the worst case.(It is almost impossible to take the largest or smallest element as the pivot.)

(d) Disadvantages:
The extra cost of generate the random number and swap the pivot with the begin or end element of the array.
Advantages:
If the data is roughly in order, the random pick of pivot will significantly decrease the probability of a unbalanced time complexity function. ($f(n) = f(m) + f(n-m) + kn$ where m is close to n) And this will improve the efficiency of the algorithm.
However if the input is totally random that is the possibility of each specific element appear at the position is equal, I guess the random pick is totally useless.
From $f(n) = f(m) + f(n-m) + kn + c$ where c is the cost of random pick.
In the worse case $cn \log n$; in the average case $c \log n$ extra time cost is needed but surely the time complexity is still $O(n^2)$ and $O(n \log n)$ respectively.

(e)
```
ithelement(arr[],beg,end,i){
        p = partition(arr[],beg,end); //position of the pivot
        if(p = i) then return p
        else if (p < i)
                then ithelement(arr[],beg,p-1,i);
                else ithelement(arr[],p+1,end,i);
}
```

If n is odd then the median is ithelement(arr[],beg,end,n/2).
If n is even then we need to find the element at n/2 and n/2 + 1 and take average.(Here n/m = quo(n,m)).
There should be better solutions for the case n is even. For example, use the partially sorted result from find the first median at n/2 position.
Let f(n) be the time complexity of ithelemet of size n input array.

$$f(n) = f(m) + kn \ (m \in [1, n])$$
$$f(n) = k \log n$$