

16:332:573

Midterm Exam I

Duration: 150 minutes, a closed book closed note exam.

Q1. Fastest 3-sum: In the lecture, we discussed $O(N^2 \log N)$ algorithm by introducing binary search to a brute-force version. I mentioned that we could achieve $O(N^2)$ using hash table, which will be covered in later part of the course. However, we can develop another version of $O(N^2)$ algorithm for 3-sum problem, without using hash table, by carefully testing all possible pairs. Your new algorithm may neither use hash table nor binary search. Develop such an algorithm, write a pseudocode, and briefly analyze the algorithm, i.e., why it is $O(N^2)$, etc. [20pt]

Sol)

```
S := an input array of length n.
sort(S);
for i=0 to n-3 do
    a = S[i];
    start = i+1;
    end = n-1;
    while (start < end) do
        b = S[start]
        c = S[end];
        if (a+b+c == 0) then
            output a, b, c;
            // Continue search for all triplet combinations summing to zero.
            end = end - 1
        else if (a+b+c > 0) then
            end = end - 1;
        else
```

```
        start = start + 1;
    end
end
end
```

- **Grading.**

- If the idea behind the pseudocode is correct, get 15pt.
- If some details are wrong, up to 5pt can be deducted.
 - E.g., for $i=0:n-3$, change start/end pointers in a loop, etc.

Q2. Union-Find Analysis: Why is Weighted Quick Union with Path Compression “better” than Quick Union with Path Compression? Why are either of these “better” than Quick Union or Quick Find?

Sol)

In WQUPC, each union-find operation can make a maximum of \lg^*N array accesses as opposed to $\log N$ array accesses in QUPC. For all practical purposes, \lg^*N has an upper bound of 5 (orders of magnitude lesser than that of $\log N$) and thus WQUPC is “better” than QUPC. Both QU & QF in the worst case can have N array access and thus worse than WQUPC and QUPC.

Q3. Recursion: A child is running up a staircase with n steps, and can hop either 1 step, 2 steps, or 3 steps at a time. Write a code (pseudocode is fine) to count how many possible ways the child can run up the stairs.

Sol)

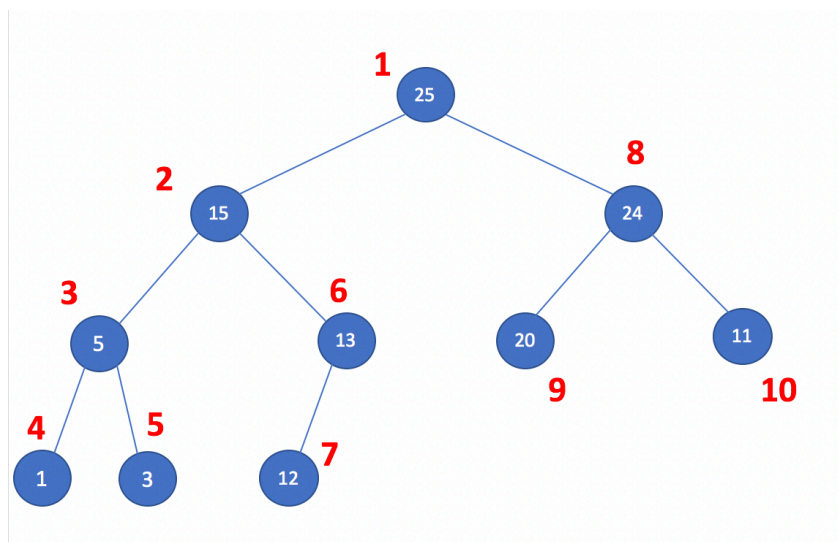
We can approach this problem from the top down in a recursive way. On the very last hop, up to the n th step, the child could have done either a single, double or triple step hop. That is, the last move might have been

a single step hop from step $n-1$, a double step hop from step $n-2$, or a triple step hop from $n-3$. The total number of ways of reaching the last step is therefore the sum of the number of ways for reaching each of the last three steps.

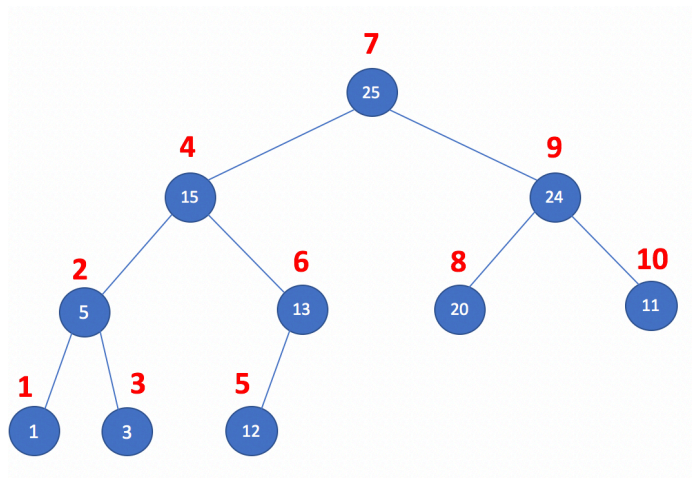
```
int countWays(int n) {
    if (n < 0) {
        return 0;
    } else if {n == 0} {
        return 1;
    } else {
        return countWays(n - 1) + countWays(n - 2) + countWays(n - 3);
    }
}
```

Q4. For the following tree, give the order of visit for the following three cases.

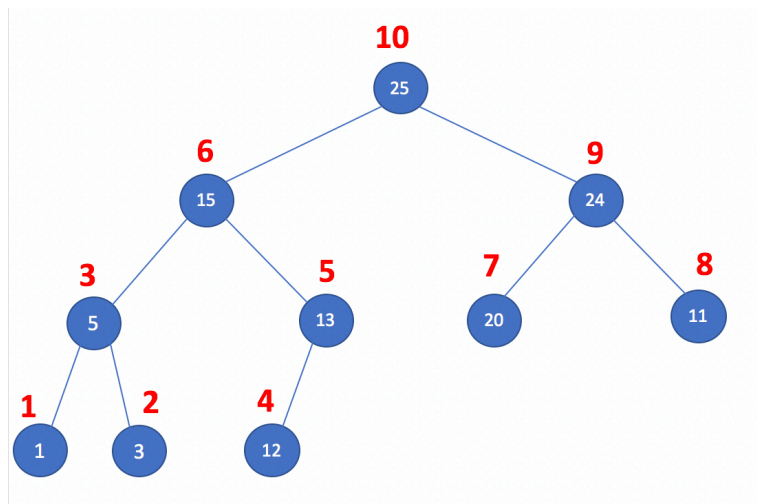
a) Pre-order



b) In-order



c) Post-order



Q5. Sorting General: Mark “T” if each of the following statement is true. Otherwise write “F”.

- a) [F] The running time of Quicksort algorithm is $O(N \log N)$.
- b) [T] When we want to sort a randomly-ordered array with distinct keys, Insertion sort uses less number of compares than that of Selection sort.
- c) [F] Merge operation in the top-down Mergesort algorithm could take N^2 in the worst case.

- d) [F] Quicksort algorithm is often faster than Mergesort algorithm because the number of recursive calls is less.
- e) [T] The average case performance of Mergesort and Quicksort algorithm is $O(N \log N)$.
- f) [T] Mergesort algorithm is an optimal algorithm in terms of # of comparisons.
- g) [F] Quicksort algorithm is an optimal algorithm in terms of # of comparisons.
- h) [T] The cutoff scheme, i.e. using insertion sort for smaller subarrays, could be useful for both Mergesort and Quicksort.
- i) [T] For a randomly-ordered array with distinct keys, Selection sort requires fewer number of exchanges compared to Insertion sort.
- j) [F] Finding top k^{th} element in a given array requires a lower bound $\sim N \log N$.

Q6. You are given two sorted arrays, A and B, where A has a large enough buffer at the end to hold B. Write a method to merge B into A in sorted order, which takes $O(N)$ time.


Note: presenting a pseudocode is fine.

Sol) The point is to use the empty space of A without shifting existing elements. If you use the merging algorithm seen in the class, you will need to shift all existing element to the right to make a room for it, which causes an inefficiency. Thus, a better version would be to start from the largest element from the back of the array A. Following is an implementation that achieves the goal.

Function merge(array a, array b, int lastA, int lastB)

```

{
    int indexA = lastA - 1;
    int indexB = lastB - 1;
    int indexMerged = lastB + lastA - 1;

    while (indexB >= 0) {
        /* end of a is > than end of b */
        if (indexA >= 0 && a[indexA] > b[indexB]) {
            a[indexMerged] = a[indexA];
            indexA--; 
        } else {
            a[indexMerged] = b[indexB];
            indexB--;
        }
        indexMerged--;
    }
}

```

Q7. Suppose that top-down Mergesort is modified to skip the call merge() whenever $a[mid] < a[mid+1]$. What is the required number of compares if an input array is already sorted? Please explain why.

Sol)

The number of compares used for an array in sorted order is linear.

Since the array is already sorted, there will be no calls to merge(). When N is a power of 2, the number of compares will satisfy the recurrence $T(N) = 2 T(N/2) + 1$, with $T(1) = 0$.

Q8. Give traces, in the style of the trace given in the lecture, showing how the keys E A S Y Q U E S T I O N are sorted with top-down mergesort and with bottom-up mergesort. In each step, please make a working set clear (the part that is being actively scanned and merged).

Sol)

i) Top-down

			a[]											
lo	m	hi	0	1	2	3	4	5	6	7	8	9	10	11
			E	A	S	Y	Q	U	E	S	T	I	O	N
0	0	1	A	E	S	Y	Q	U	E	S	T	I	O	N
0	1	2	A	E	S	Y	Q	U	E	S	T	I	O	N
3	3	4	A	E	S	Q	Y	U	E	S	T	I	O	N
3	4	5	A	E	S	Q	U	Y	E	S	T	I	O	N
0	2	5	A	E	Q	S	U	Y	E	S	T	I	O	N
6	6	7	A	E	Q	S	U	Y	E	S	T	I	O	N
6	7	8	A	E	Q	S	U	Y	E	S	T	I	O	N
9	9	10	A	E	Q	S	U	Y	E	S	T	I	O	N
9	10	11	A	E	Q	S	U	Y	E	S	T	I	N	O
6	8	11	A	E	Q	S	U	Y	E	I	N	O	S	T
0	5	11	A	E	E	I	N	O	Q	S	S	T	U	Y
			A	E	E	I	N	O	Q	S	S	T	U	Y

ii) Bottom-up

			a[]											
lo	m	hi	0	1	2	3	4	5	6	7	8	9	10	11
			E	A	S	Y	Q	U	E	S	T	I	O	N
0	0	1	A	E	S	Y	Q	U	E	S	T	I	O	N
2	2	3	A	E	S	Y	Q	U	E	S	T	I	O	N
4	4	5	A	E	S	Y	Q	U	E	S	T	I	O	N
6	6	7	A	E	S	Y	Q	U	E	S	T	I	O	N
8	8	9	A	E	S	Y	Q	U	E	S	I	T	O	N
10	10	11	A	E	S	Y	Q	U	E	S	I	T	N	O
0	1	3	A	E	S	Y	Q	U	E	S	I	T	N	O
4	5	7	A	E	S	Y	E	Q	S	U	I	T	N	O
8	9	11	A	E	S	Y	E	Q	S	U	I	N	O	T
0	3	7	A	E	E	Q	S	S	U	Y	I	N	O	T
0	7	11	A	E	E	I	N	O	Q	S	S	T	U	Y
			A	E	E	I	N	O	Q	S	S	T	U	Y

Q9. Quicksort: Let's assume basic quicksort algorithm based on a single pivot value. Explain when the quicksort algorithm will show the best performance and in which case the quicksort algorithm will show the worst performance?

Sol) The performance of quicksort algorithm will be governed by how well we choose pivot value in each step. On average, if we successfully choose median values as our pivots, the algorithm will take $O(N \log N)$ time, which is the best case. In contrast, if our pivot values are either minimum or maximum values in a given subarray, the algorithm will show the worst case performance, i.e., $O(N^2)$.

Q10. Recall Dijkstra's 3-way partitioning algorithm. For the following input array, write a resulting array after the partitioning operation. How many comparisons are required?

K A U K R K C Y W B N K Q L P V D G K Z

Sol) The algorithm was as follows. Let v be partitioning item $a[lo]$. Scan l from left to right.

- ($a[i] < v$): exchange $a[l]$ with $a[i]$; increment both l and i .
- ($a[i] > v$): exchange $a[gt]$ with $a[i]$; decrement gt .
- ($a[i] == v$): increment i

K A U K R K C Y W B N K Q L P V D G K Z
A K U K R K C Y W B N K Q L P V D G K Z
A K Z K R K C Y W B N K Q L P V D G K U
A K K K R K C Y W B N K Q L P V D G Z U
A K K K G K C Y W B N K Q L P V D R Z U
A G K K K K C Y W B N K Q L P V D R Z U
A G C K K K K Y W B N K Q L P V D R Z U
A G C K K K K D W B N K Q L P V Y R Z U
A G C D K K K K W B N K Q L P V Y R Z U
A G C D K K K K V B N K Q L P W Y R Z U
A G C D K K K K P B N K Q L V W Y R Z U
A G C D K K K K L B N K Q P V W Y R Z U
A G C D K K K K Q B N K L P V W Y R Z U
A G C D K K K K K B N Q L P V W Y R Z U
A G C D B K K K K K N Q L P V W Y R Z U