Tutorial

November 1, 2021

PS1-P1

- Bubble Sort.
  - Find the smallest element in each iteration;
  - Time Complexity: $O(n^2)$.

PS1-P2

- Polynomial Evaluation: $\sum_{k=0}^{n} c_k x^k$.
  - $\sum_{k=i}^{n} c_k x^{k-i}$;
  - Time Complexity: $O(n)$.

PS1-P3

- $(\log n)^{\log n}$ and $2^{(\log n)^2}$: take logarithm on both side.

PS1-P4

- $\log(\log^* n), \log^* n$ and $\log^*(\log n)$: $\log^*(\log n) = log^* n - 1$, $\log(\log^* n) = o(\log^* n)$ ($\log^* n \to +\infty$ when n tends to infinity).
- Stirling's formula: $n! \sim \frac{1}{\sqrt{2\pi n}} \left(\frac{n}{e}\right)^n$:
  - $\log(n!) \sim n \log n$, $(\log n)! = o\left((\log n)^{\log n}\right)$.

PS1-P5

- Implement two stacks in one array:
  - Initialization: $p1 = 0, p2 = n + 1$;
  - Push of stack1(x) : $p1 \leftarrow p1 + 1, A[p1] \leftarrow x$;
  - Push of stack2(x) : $p2 \leftarrow p2 - 1, A[p2] \leftarrow x$;
  - Pop of stack1 :$p1 \leftarrow p1 - 1$;
  - Pop of stack1 :$p2 \leftarrow p2 + 1$.

PS1-P6

- Implement stack via two queues P, Q:
    - $push(x)$: $P.enqueue(x)$;
    - $pop(x)$:
        - repeat $Q.enqueue(P.dequeue())$ until the size of P is exactly 1;
        - record the return value with $P.dequeue()$;
        - repeat $P.enqueue(Q.dequeue())$ until Q is empty.
- push: $O(1)$, pop: $O(n)$.

PS1-P6'

- Implement queue via two stacks P, Q:
  - *enqueue*(x): push x to stack P;
  - *dequeue*(x): If T is empty, pop element from stack S and push it to stack T repeatedly, until stack S is empty. After above operations, T is non-empty, and we will pop element from T, and return it.
- push: $O(1)$, pop: $O(n)$, amortized $O(1)$.

PS1-P7

- Adding and randomized removing.
  - initialization: $size = 0$;
  - $add(x)$: $A[size] \leftarrow x$, $size \leftarrow size + 1$;
  - $remove(x)$: $i \leftarrow random(size) - 1$, $size \leftarrow size - 1$, swap($A[i]$,$A[size]$).

PS2-P1

- Linked list reversal.
- Exclusive-Or Linked List: $x.next = x.np \oplus x.prev$ (be careful with updates of $x.np$).

PS2-P2

- Max-stack:
  - push(x): If S is nonempty, let $y = S.pop()$, then push y to the stack S, and lastly push $\{x, max(y.second, x)\}$ to the stack S; otherwise, push $\{x, x\}$ to the stack S.
  - pop(): Let $y = S.pop()$, return $y.first$.
  - max(): Let $y = S.pop()$, push y to the stack S, and lastly return $y.second$.

PS2-P3

- Infix expression to postfix expression:

Algorithm

Create stack S initialized to empty. Set $pri[!] > pri[x] > pri[+]$.

---

for $i = 0$ **to** n **do**
  if $A[i]$ is digit **then** Print(A[i])
  else
    while S is not empty and $pri\,[S.top()] \geq pri\,[A[i]]$ **do** $Print\,(S.pop())$
    $S.push(A[i])$
while S is not empty **do** $Print\,(S.pop())$

PS2-P4

- Application of Master's Theorem.

PS2-P5

- Duplicate removal: sort in $O(n \log n)$ and remove in linear time.

PS2-P6

- Count the number of inversions in $O(n \log n)$: D&C.
- The number of inversions equal to the number of swaps in insertion sort.

PS3-P1

- $T(n) = 2T\left(\frac{n}{2}\right) + n$: substitution method;
- $T(n) = T(n-2) + T(n/2) + n$: appropriate grouping and substitution method.

PS3-P2

- $T(n) = T(\alpha) + T(n-\alpha) + cn$ vs $T(n) = T(\alpha n) + T((1-\alpha)n) + cn$.
- Recursion Tree.

PS3-P3

- Square a binary number of length n in $O(n^{\log 3})$;
- $(2^k a + b) = 2^{2k} a^2 + b^2 + 2^{k+1} ab = 2^{2k} a^2 + b^2 + 2^k (a^2 + b^2 - (a - b)^2)$.

PS3-P4

- Find x in a monotone sequence with unknown length of finite numbers.
- binary search with a twist.

PS3-P5

- Find the majority via comparison.
- Randomized Algorithm. Expected $O(n)$ times of comparison.
- Voting Algorithm.
- D&C in $O(n \log n)$ and pairing stategy in $O(n)$.

PS3-P6

- Find maximum subarray sum in $O(n)$.
- D&C technique: maintain maximum subarray sum containing left border and right border.
- Simpler algorithm: Let $S_i = \sum_{k=1}^{i} A_i$. It suffices to find the minimum of $S_0, S_1, \ldots, S_{i-1}$ for all $1 \leq i \leq n$.
- Challenge: Find $\sum_{1 \leq i \leq j \leq n} mss([A_i, A_{i+1}, \ldots, A_j])$ in $O(n \log^2 n)$ time.

PS3-P7

- Find k-th largest element in a binary max-heap in $O(k \log k)$ time.
- Find all possible candidates in each iteration.