0. Practice Exercise (do not turn in): Solving Recurrences

Give a tight asymptotic bound for the following recurrences. In each case explain the technique you use and why your answer is correct. For all these problems $T(1) = 1$. (Hint: In some cases it’s useful to write out the recursion tree.)

(a) $T(n) = 2T(\lfloor n/2 \rfloor) + 1$
(b) $T(n) = 3T(\lfloor n/2 \rfloor) + n \lg n$.
(c) $T(n) = 3T(\lfloor n/2 \rfloor) + n^3$
(d) $T(n) = T(\lceil \sqrt{n} \rceil) + 1$
(e) $T(n) = n^{2/3} T(\lfloor n^{1/3} \rfloor) + n$.

(25 pts) 1. Part of a Sorted Array

Give a linear time algorithm that takes as input a list of $n$ distinct numbers $a_1, a_2, \ldots, a_n$, and outputs the smallest one, the second smallest one, the fourth smallest one, $\cdots$. In other words, if $b_1 < b_2 < \ldots < b_n$ is the set of elements in the input list after they have been sorted, then the output is a sequence $b_1, b_2, b_4, \ldots, b_{2^k}$, where $2^k$ is the greatest power of 2 that is at most $n$.

For example, if the input sequence is $[7, 6, 5, 4, 3, 2, 1, 0]$ (or any permutation of these numbers) then the algorithm would produce $[0, 1, 3, 7]$.

As usual, you should prove your algorithm is correct, and prove your bound (in this case linear) on the running time.
(25 pts) 2. BrilliantSort

Consider the following sorting algorithm $\text{BSort}(A, i, k)$ which takes as input an array $A$ and sorts (in place) the elements $A[i]$, $A[i+1]$, ..., $A[i+k-1]$. That is, it sorts the subarray of $A$ starting at $i$ of length $k$, which is always a power of 2, and is at least 2. Here is the algorithm expressed in pseudo-C:

```pseudo
BSort(A, i, k) {
    if (k=2) {
        if (A[i] > A[i+1]) swap (A[i], A[i+1]);
    } else {
        for (j=0; j<2; j++) {
            BSort(A, i, k/2);
            BSort(A, i + k/2, k/2);
            BSort(A, i + k/4, k/2);
        }
    }
}
```

(a) Assuming $A$ is an array whose length is $n \geq 2$, which is a power of 2, prove that $\text{BSort}(A, 0, n)$ will sort $A$.

(b) Prove that the number of comparisons involving $A$ done by $\text{BSort}(A, 0, n)$ is dependent only on $n$ and not on the contents of $A$.

(c) Write and solve the recurrence for $T(n)$ the number of comparisons involving $A$ done by $\text{BSort}(A, 0, n)$.

(25 pts) 3. A Little Bit Disordered


For each element $i$ (in increasing order from 1 up to $n-2$) $A[i]$ is swapped with $A[i+r]$, where $r$ is randomly chosen (on each step) from $\{0, 1, 2\}$.

(a) As a function of $n$, how many possible distinct permutations of $A[]$ could be generated? (This is not a big-oh question.)

(b) Give a lower bound on the number of comparisons required in the worst cast to sort $A[]$. What is that lower bound? (This is not a big-oh question.)

(c) Give a comparison-based algorithm that sorts such permuted arrays using at most $2n - 4$ comparisons. [Hint: it may be useful to prove a lemma limiting how far an element can be to the left of where it belongs.]
4. **Programming: Sorting with Block Swaps**

A *block swap* takes a pair of neighboring blocks of an array $A$ of the same size and exchanges them. For example if $A = [0, 1, 3, 2, 4, 6, 5, 7]$ in one block swap we could get $[4, 6, 5, 7, 0, 1, 3, 2]$, or $[0, 3, 1, 2, 4, 6, 5, 7]$, or $[2, 4, 6, 0, 1, 3, 5, 7]$, but not $[7, 1, 3, 2, 4, 6, 5, 0]$.

Write a program which takes as input a permutation of the numbers $0, 1, \ldots, n - 1$. View these numbers as the contents of a zero-based array $A$. Your program will compute a sequence of pairs $(i, \ell)$, which represents swapping the block $[A[i], A[i + 1], \ldots, A[i + \ell - 1]]$ with $[A[i + \ell], A[i + \ell + 1], \ldots, A[i + 2\ell - 1]]$. This sequence of swaps should sort the array $A$.

The number of swaps generated must be at most $4n$. The running time bound will be 5 seconds. *Please include a comment at the start of your program explaining your algorithm, and why it does only $O(n)$ swaps.*

Details on how to submit, grading policy, etc., will be on the course website and piazza soon.

**INPUT:** The first line contains $n$, which is at most $3 \times 10^4$. The second line consists of the numbers $a_0, a_1, \ldots, a_{n-1}$ separated by blanks. These numbers will be a permutation of $0, 1, \ldots, n - 1$.

**OUTPUT:** The first line of output is two space separated integers $i_1 \ell_1$, which describe the first block swap. Subsequent swaps are on subsequent lines. The last swap is followed by a terminating line consisting of two zeros.

For example, if the input is:

```
5
0 4 1 3 2
```

then the output could be

```
2 1
1 2
3 1
0 0
```