Assignment 11, Data-Structures

Oleg Sivokon

<2016-03-06 Sun>

Contents

1 Problems

1.1 Problem 1

Count the number of compare and copy operations required to sort the two sequences given below using insertion sort:

1. $\frac{n}{2}, \frac{n}{2} - 1, \ldots, 2, 1, n, n-1, \ldots, \frac{n}{2}, \frac{n}{2} - 1.$
2. $n, 1, n-1, 2, \ldots, \frac{n}{2} + 2, \frac{n}{2} - 1, \frac{n}{2} + 1, \frac{n}{2}.$

1.1.1 Answer 1

Assuming we sort in ascending order, observe that the our task is to repeat the same operation twice (viz. to reverse two sorted arrays of the size exactly half of n.) Reversing individual arrays will require in case of one-length array 0 `swap` operations, in case of two-length array, 1 `swap` operation, and when we go further, we would need to do the same amount of work we did in the case of $n-1$, and then need to do $n-1$ more swaps to bring the first element of the original input to the back.

This gives the recurrence (for reversing the array):

$$\begin{aligned}
R(1) &= 0 \\
R(2) &= 1 \\
R(3) &= R(2) + (3-1) = 3 \\
R(4) &= R(3) + (4-1) = 6 \\
&\ldots \\
R(n) &= R(n-1) + n - 1 = \sum_{i=0}^{n} i = \frac{n(n+1)}{2}\ .
\end{aligned}$$

Hence the total amount of `swap` operations needed to sort the given array is $T(n) = 2R(\frac{n}{2}) = n(n+1)$.

It is easy to see that the asymptotic complexity of $T(n)$ is $O(n^2)$ since it is definitely bounded below by n^2 (since $n \times n < n \times (n+1)$.)

1.1.2 Answer 2

Assuming we sort in ascending order, in the first step we make one comparison and swap. In the next step the first element will stand in its place, while the second element will need to move two positions to the end. The one before last element will need to move one position back.

The same will happen once we increment further. I.e. now two elements will stand in their place, but now the largest element will need to move two positions to the end, and so will the second largest element. The third smalles element will need to move one position to the front.

Let now $T(n)$ denote the number of `swap` operations we perform for any given n, then:

$$\begin{aligned}
T(2) &= 1 \\
T(4) &= T(2) + 2 + 1 \\
T(6) &= T(4) + 2 + 2 + 1 \\
T(8) &= T(6) + 2 + 2 + 2 + 1 \\
&\dots \\
T(n) &= T(n-2) + n - 1 \; .
\end{aligned}$$

This recurrence is easily recognizable as being just $(n-1)^2$. Hence, as expected, insertion sort requires roughly quadratic number of swaps, i.e. $O(n^2)$.

1.2 Problem 2

Given a sorted array $A[1 \ldots n]$ where all elements are unique natural numbers:

1. Write a predicate that asserts whether the given array is dense (has no gaps) in time $\Theta(1)$.
2. Write a predicate that given that A is sparse finds the element v which doesn't apear in A but is smaller than its largest element and is greater than its smallest element.

1.2.1 Answer 3

Real code (compiled in C99):

Note that you will need the support code located in this derictory

```
bool is_sparse(const array* sorted) {
    int* first = (int*)sorted->elements[0]->val;
    int* last = (int*)sorted->elements[sorted->length - 1]->val;

    return (int)*last - (int)*first >= sorted->length;
}

size_t binsearch_missing(const array* sparse) {
    size_t start = 0, end = sparse->length / 2;
    array* cut = slice(sparse, start, end);

    while (end - start > 1) {
        if (is_sparse(cut)) {
            end -= (end - start) / 2;
        } else {
            start = end - 1;
            end += (sparse->length - end) / 2;
        }
        free_array(cut);
        cut = slice(sparse, start, end);
    }
    return end + 1;
}

void report(array* tested, char* message) {
    printf(message, to_string((printable*)tested));
    if (!is_sparse(tested)) {
```

```c
        printf("Array is dense.\n");
    } else {
        printf("Array is sparse.\n");
        size_t missing = binsearch_missing(tested);
        printf("The first gap is at: %d\n", (int)missing);
    }
}

int main() {
    report(make_sparse_sorted_array(10, 13, 7, int_element_generator),
           "Created sparse array: %s.\n");
    return 0;
}
```

```
Created sparse array: [13, 19, 22, 28, 34, 40, 43, 47, 49, 52].
Array is sparse.
The first gap is at: 2
```

1.3 Problem 3

1.3.1 Answer 3

1.4 Problem 4

Show example of a function f satisfying $f(n) \neq \Omega(n)$ and $f(n) \neq (n)$.

1.4.1 Anwser 4