COMP10002 Foundations of Algorithms

Workshop Week6

Wenbin Cao

August 29, 2019

GitHub Repo: https://github.com/AlanChaw/COMP10002-FoA

Recap

Chapter 12 - Measuring Performance

Chapter 12 - Quicksort

Chapter 7 - String

Measuring Performance

- Correcrtness

- Efficiency

  - Time
  - Space

Defining Efficiency

- $O(g(n))$ is a *set* of functions, including all functions that have the same or smaller growth rate.

  - $f_1(n) = 2n$. Then $f_1(n) \in O(n)$.
  - $f_2(n) = 3n^2 + 2n + 3$. Then $f_2(n) \in O(n^2)$
  - $f_3(n) = 15n + 27\sqrt{n} - e \log n$. Then $f_3(n) \in O(n)$

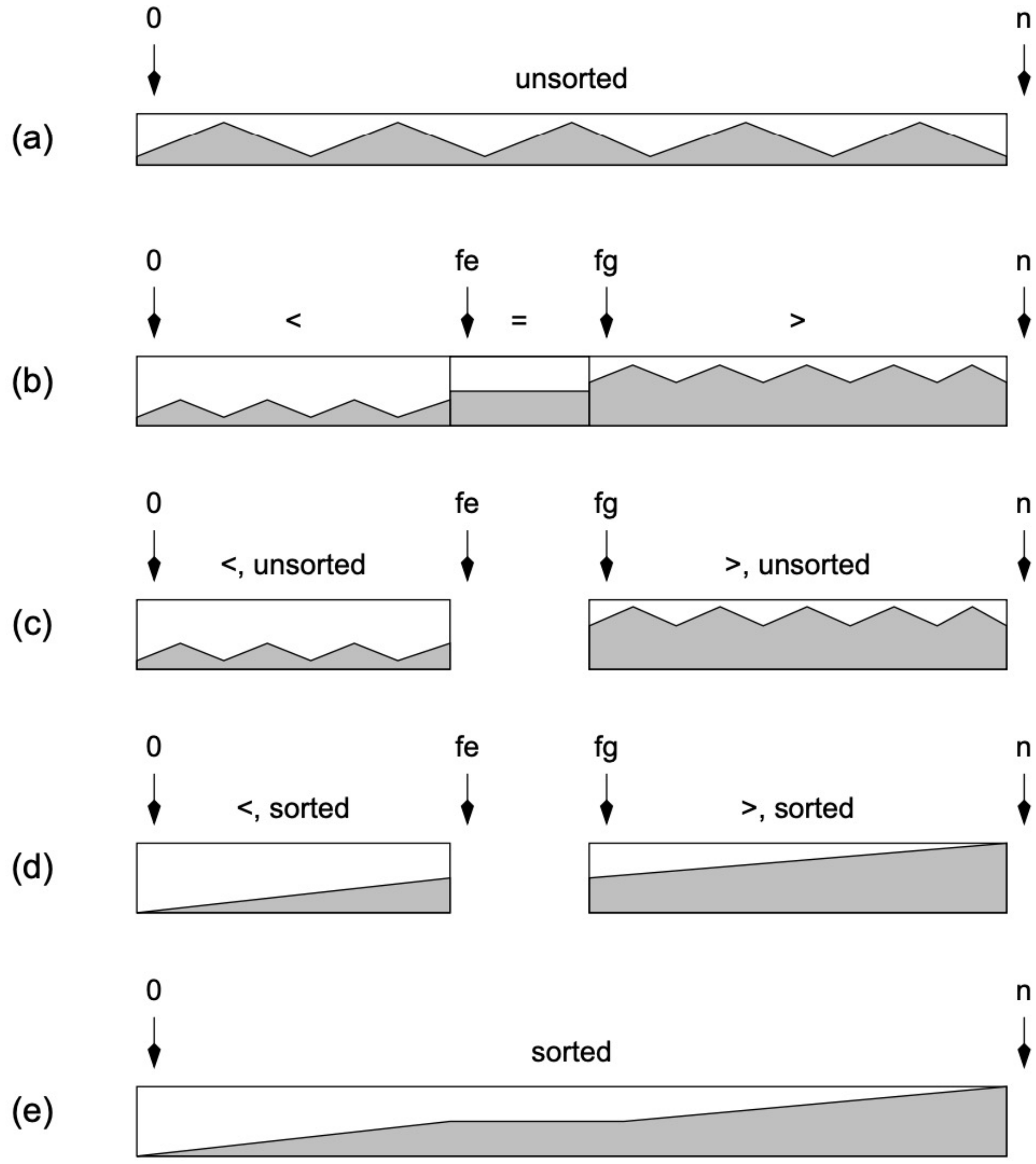
- It is usual to make use of the *simplest* such $g(n)$ function

- Allows the growth rate of function can be compared

Time Complexity - Insertion Sort

```
void sort_int_array(int A[], int n) {
    int i, j;
    for (i=1; i<n; i++) {
        for (j=i-1; j>=0 && A[j+1]<A[j]; j--) {
            int_swap(&A[j], &A[j+1]);
        }
    }
}
```

Quicksort



Unsorted

Partitioned

Unsorted sections

Recursive calls

Sorted

Partition Example

7 4 6 3 1 2 5

Pivot = 3

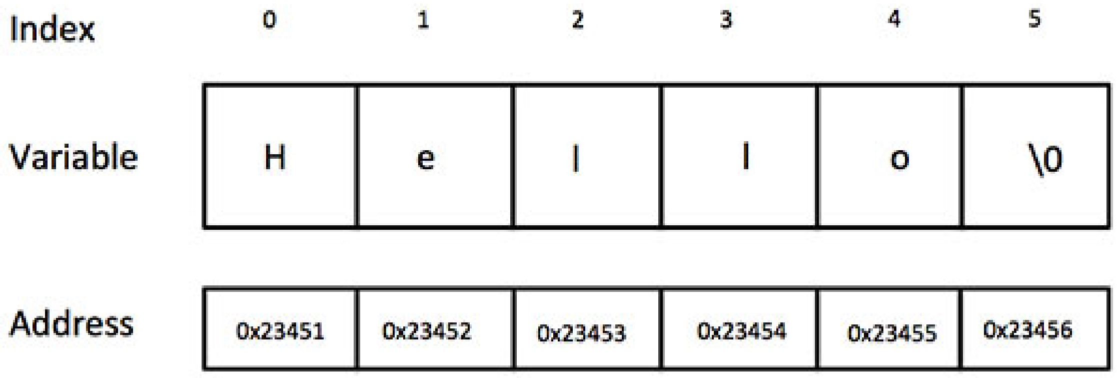
Time Complexity - Quick Sort

- $O(n \log n)$ time, on average
- $O(n^2)$ in the worst case, but almost impossible.

String

- In C, there is no pre-defined string type, instead, we use char [].

```
char *p = "Hello";   /* cannot be modified */
/*
char p[6] = {'H','e','l','l','o','\0'};
char p[] = "hello";
*/
printf("%s\n", p);
printf("%c\n", *p);
printf("%c\n", *(p+1));
printf("%c\n", p[2]);
```



```
hello
h
e
l
```

Array of Strings

Example:

```
char *words[10] = {"Algorithms", "are", "fun"};
int i;
for (i = 0; words[i] != NULL; i++) {
    printf("%s\n", words[i]);
}
```

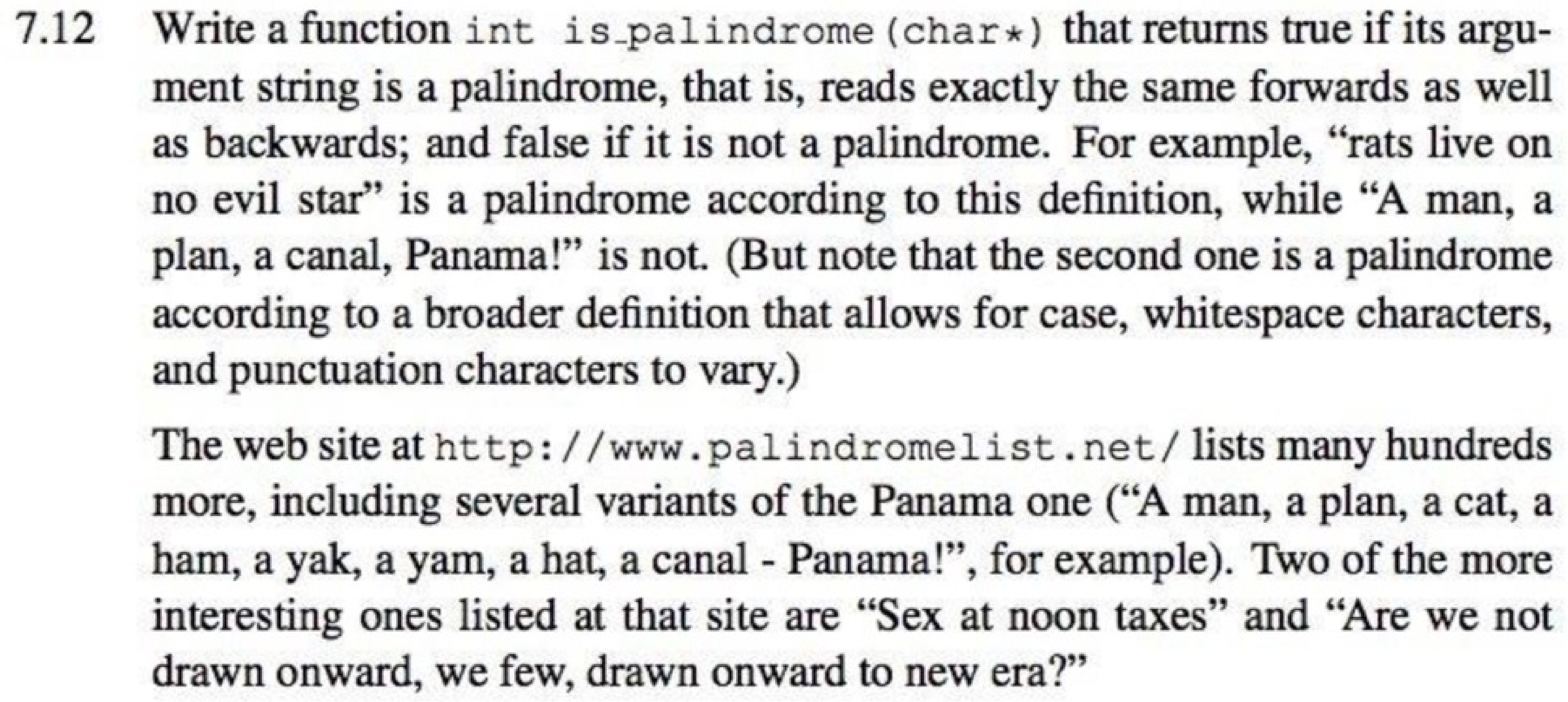
Output:

```
Algorithms
are
fun
```

Read one line of input into a string

```c
int
read_line(char str[], int max) {
    int n=0, c;
    while ((c=getchar()) != EOF && c!='\n') {
        str[n++] = c;
        if (n==max-1) {
            str[n++] = '\0';
            return 1;
        }
    }
    if (c==EOF && n==0) {
        return 0;
    }
    str[n++] = '\0';
    return 1;
}
```

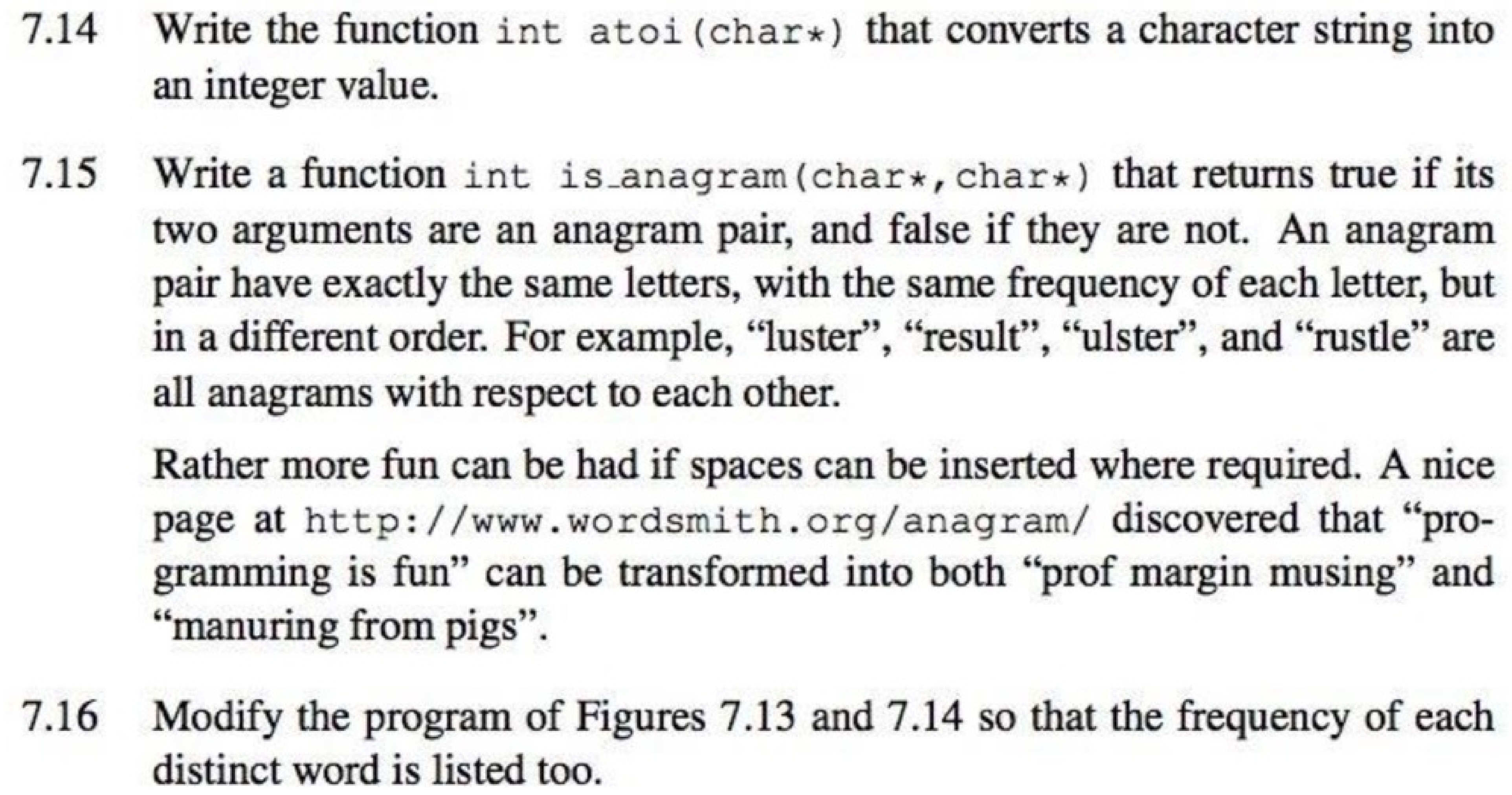
Exercise



7.12 Write a function `int is_palindrome(char*)` that returns true if its argument string is a palindrome, that is, reads exactly the same forwards as well as backwards; and false if it is not a palindrome. For example, “rats live on no evil star” is a palindrome according to this definition, while “A man, a plan, a canal, Panama!” is not. (But note that the second one is a palindrome according to a broader definition that allows for case, whitespace characters, and punctuation characters to vary.)

The web site at `http://www.palindromelist.net/` lists many hundreds more, including several variants of the Panama one (“A man, a plan, a cat, a ham, a yak, a yam, a hat, a canal - Panama!”, for example). Two of the more interesting ones listed at that site are “Sex at noon taxes” and “Are we not drawn onward, we few, drawn onward to new era?”

Exercise



7.14 Write the function `int atoi(char*)` that converts a character string into an integer value.

7.15 Write a function `int is_anagram(char*,char*)` that returns true if its two arguments are an anagram pair, and false if they are not. An anagram pair have exactly the same letters, with the same frequency of each letter, but in a different order. For example, "luster", "result", "ulster", and "rustle" are all anagrams with respect to each other.

Rather more fun can be had if spaces can be inserted where required. A nice page at `http://www.wordsmith.org/anagram/` discovered that "programming is fun" can be transformed into both "prof margin musing" and "manuring from pigs".

7.16 Modify the program of Figures 7.13 and 7.14 so that the frequency of each distinct word is listed too.

Exercise 7-14

For 7-14, you may use function isdigit() in <string.h> to check if a character is a digit:

```c
#include <string.h>

char a = 3;
isdigit(a);    /* return 1 for digit, 0 for not digit */
```

Convert a character to digit:

```c
char three = '3';
int num = three - '0';  /* num = 3 now */
```