Ch.4

4.1 Write out the code for the earlier sum function.

#include <stdio.h>

#include <stdlib.h>

/\*\*4.1 Write out the code for the earlier sum function.\*\*/

int sum(int arr[],int len);

int main()

{

int ret=0;

int arr[]={3,4,5,5,6,7};

int len=sizeof(arr)/sizeof(arr[0]);

ret= sum(arr,len);

printf("sum=%i",ret);

return 0;

}

int sum(int arr[],int len)

{

int i=0;

if(len==0)

{

return 0;

}

else

{

return arr[len - 1] + sum(arr, len - 1);

}

}

4.2 Write a recursive function to count the number of items in a list.

int count(int arr[], int len)

{ if (len == 0) { return 0; }

else {

return 1 + count(arr, len - 1);

}

}

4.3 Find the maximum number in a list.

int Find\_max(int arr[], int len, int max) {

if (len == 0) {

return max;

}

if (arr[len - 1] > max) {

max = arr[len - 1];

}

return Find\_max(arr, len - 1, max);

}

4.4 Remember binary search from chapter 1? It’s a divide-and-conquer algorithm, too. Can you come up with the base case and recursive case for binary search?

int binary\_search(int arr[],int first ,int last,int Snumber){

int midpoint= (first+last)/2 ;

// Recursive cases

if(Snumber==arr[midpoint]){

return midpoint;

}

else if(Snumber>arr[midpoint]){

return binary\_search(arr,midpoint+1 ,last,Snumber);

}

else if(Snumber>arr[midpoint]) {

return binary\_search(arr,first,midpoint-1,Snumber);

}

// Base case: search range is empty

if (first > last) {

return -1; // Element is not found

}

}

How long would each of these operations take in Big O notation?

4.5 Printing the value of each element in an array.

O(n)

4.6 Doubling the value of each element in an array.

O( n)

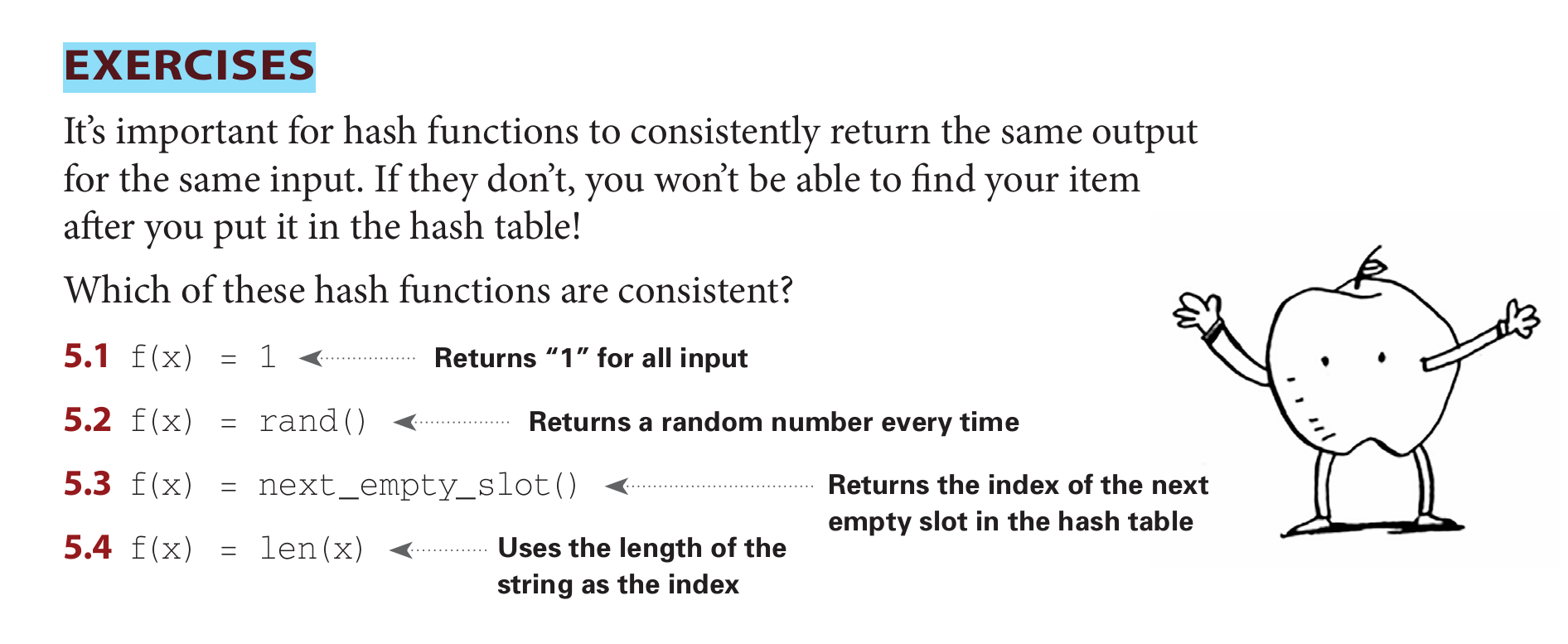
4.7 Doubling the value of just the first element in an array.

O(1)

4.8 Creating a multiplication table with all the elements in the array. So if your array is [2, 3, 7, 8, 10], you first multiply every element by 2, then multiply every element by 3, then by 7, and so on.

O(n^2)

Ch.5



5.1 ) consistent

5.2) not consistent

5.3) not consistent

5.4) consistent

It’s important for hash functions to have a good distribution. They should map items as broadly as possible. The worst case is a hash function that maps all items to the same slot in the hash table. Suppose you have these four hash functions that work with strings:

1. Return “1” for all input
2. Use the length of the string as the index.
3. Use the first character of the string as the index. So, all strings starting with a are hashed together, and so on.
4. Map every letter to a prime number: a = 2, b = 3, c = 5, d = 7, e = 11, and so on. For a string, the hash function is the sum of all the characters modulo the size of the hash. For example, if your hash size is 10, and the string is “bag”, the index is 3 + 2 + 17 % 10 = 22 % 10 = 2.

For each of these examples, which hash functions would provide a good distribution? Assume a hash table size of 10 slots.

5.5 A phonebook where the keys are names and values are phone numbers. The names are as follows: Esther, Ben, Bob, and Dan.

ASN: Hash function algorithm C,D

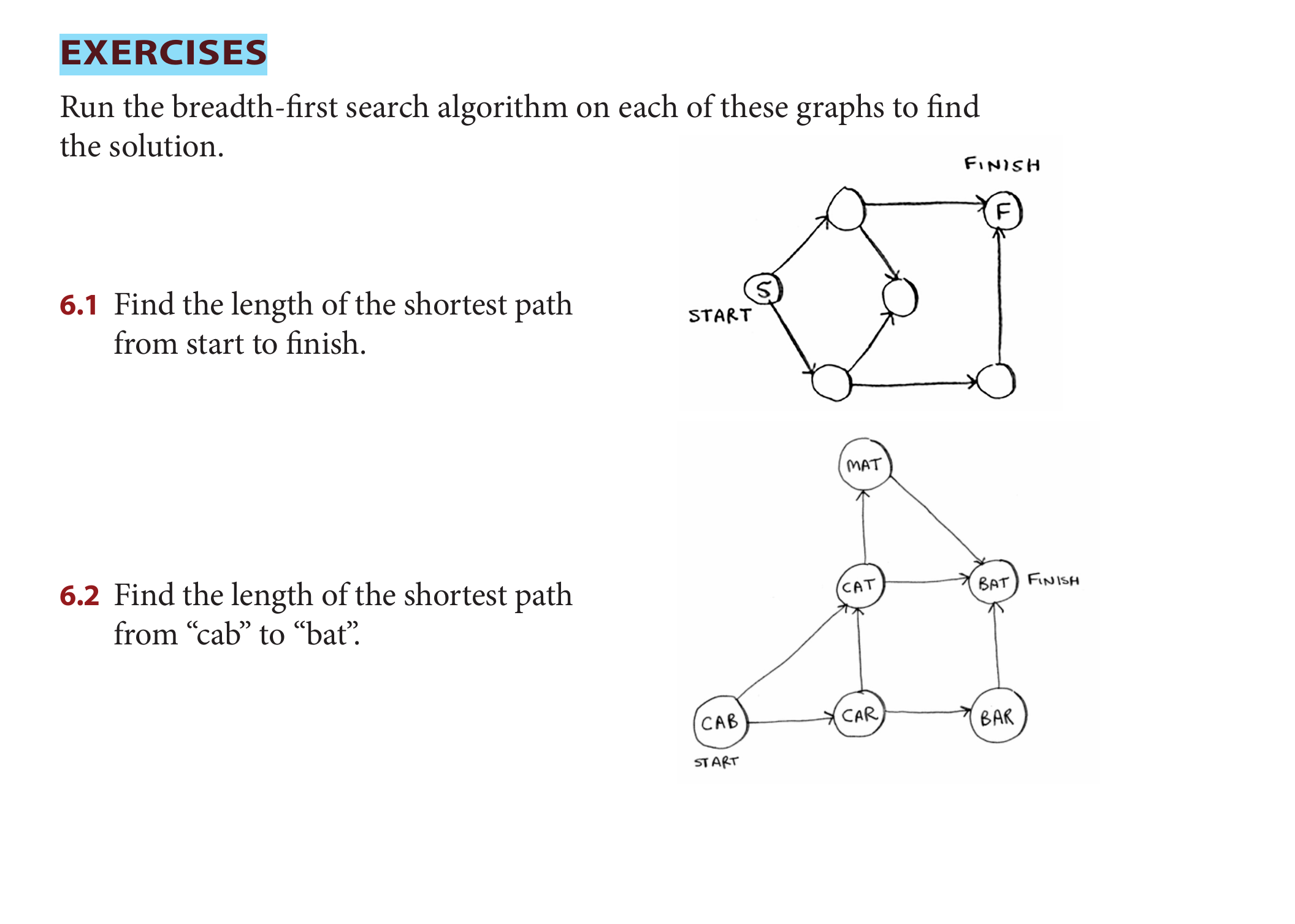
5.6 A mapping from battery size to power. The sizes are A, AA, AAA, and AAAA.

ASN: Hash function algorithm B,D

5.7 A mapping from book titles to authors. The titles are Maus, Fun Home, and Watchmen.

ASN: Hash function algorithm A,C,D

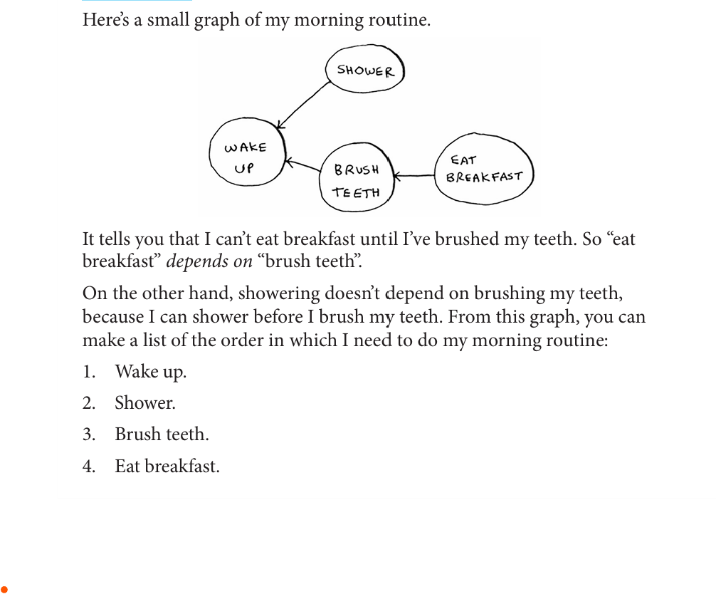
Ch.6

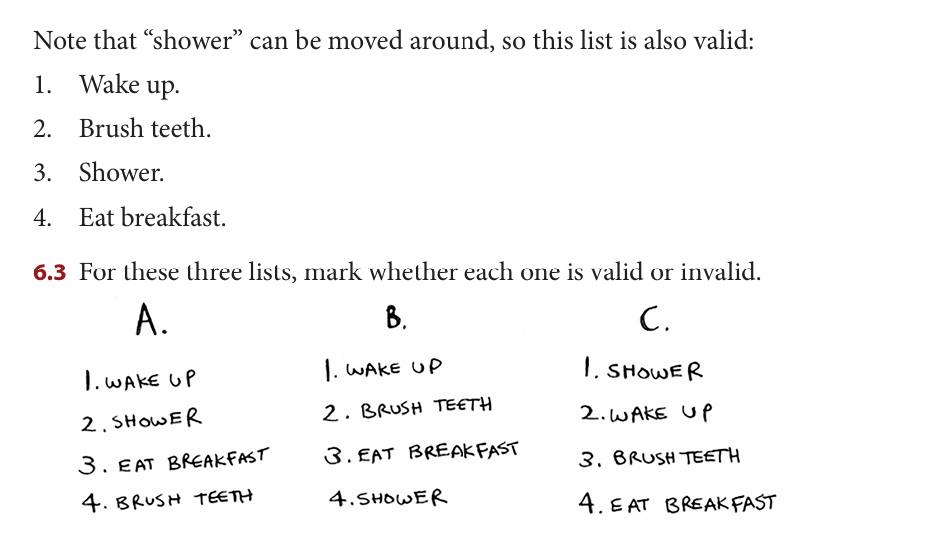




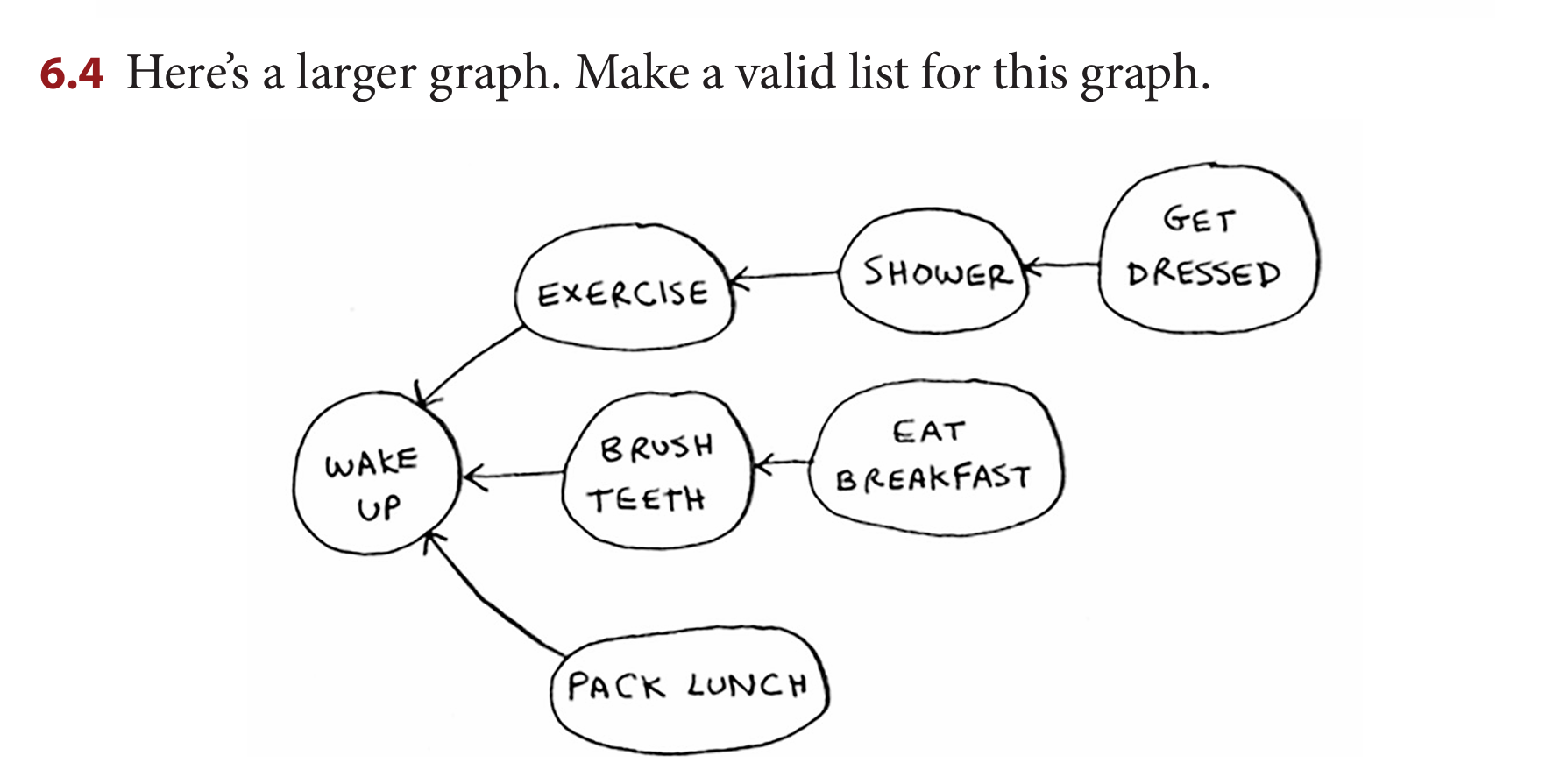
6.1 ) shortest path is 2

6.2) shortest path is 2

6.3)



1. Wrong
2. True
3. Wrong



1)wake up

2)exercise

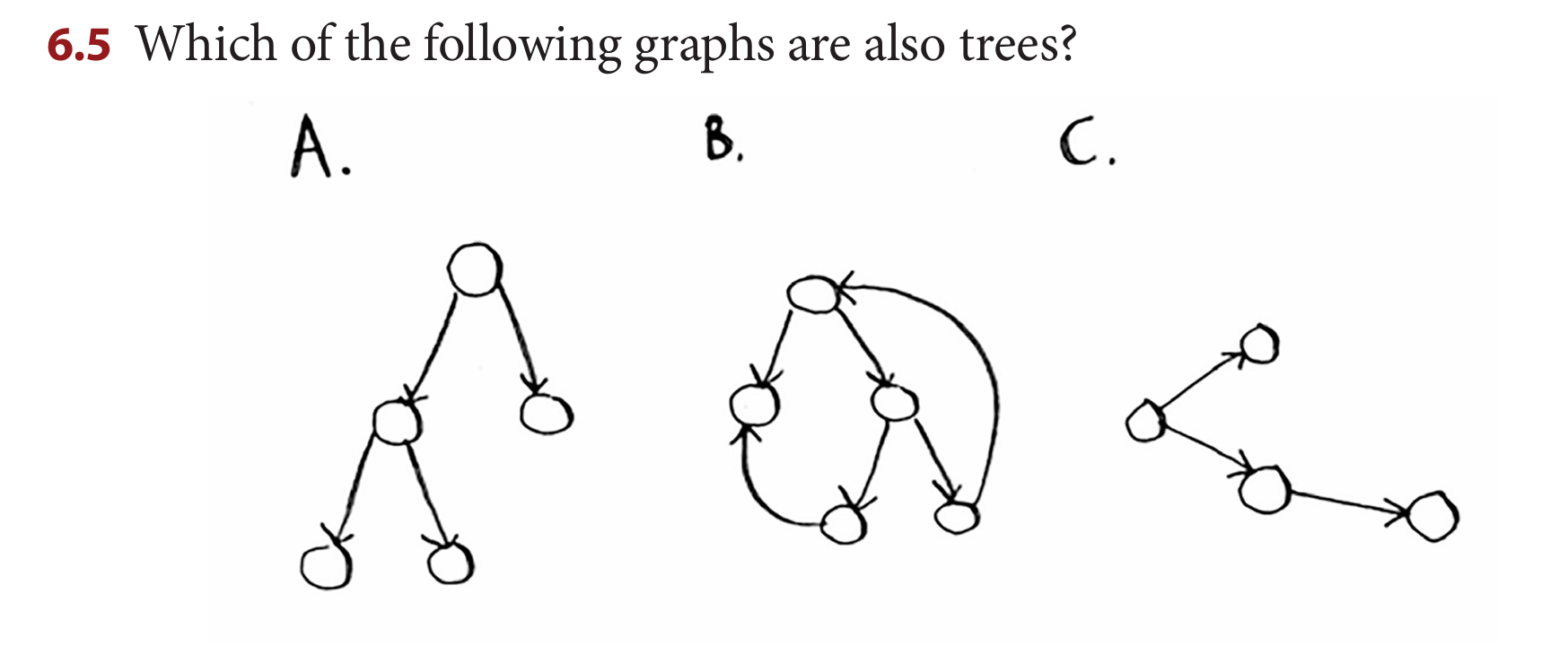
3)shower

4)get dressed

5)brush teeth

6) eat breakfast

7)pack lunch



A) Tree

b) Not a tree

c) Tree