5CCS2FC2: Foundations of Computing II

Tutorial Sheet 10

Solutions

10.1 An exam comprises two separate parts **Section A** and **Section B**. Section A comprises 10 multiple choice questions, each worth 5 marks and section B comprises 7 questions, each worth 6 marks.

On average, students answered 6.5 questions correctly on section A and 5 questions correctly on section B. However, there was a greater deviation in the number of correctly answered questions for Section A with a variance of 1.1, compared with a variance of 0.6 for Section B.

(i) What is the expectation and variance among all the exam marks?

(ii) The students also recieve 8 marks from a coursework assessment that is marked 'pass/fail', in which all students passed. How does this affect the expectation and variance of the combined coursework+exam grade?

SOLUTION:

(i) Given the information above, let

X = number of questions answered correctly in Section A

Y = number of questions answered correctly in Section B

so that

$$\mathbf{E}[X] = 6.5 \qquad \text{and} \qquad \mathbf{E}[Y] = 5$$

The final exam mark is

$$E[5X + 6Y] = 5E[X] + 6E[Y] = 5(6.5) + 6(5) = 62.5$$

and the variance is

$$Var(5X+6Y) = 25Var(X)+36Var(Y) = 25(1.1)+36(0.6) = 49.1$$

(ii) Since the expectation in linear we have that

$$E[5X + 6Y + 8] \;=\; (5E[X] + 6E[Y]) + 8 \;=\; 70.5$$

However, the variance remains indifferent to a constant shift, since

$$Var(5X + 6Y + 8) \;=\; 25Var(X) + 36Var(Y) \;= 49.1$$

as before.

10.2 Apply the Bucket Sort algorithm to the following list of numbers:

(i) 0.84, 0.98, 0.64, 0.63, 0.79, 0.58, 0.62, 0.26, 0.15, 0.33

(ii) 0.01, 0.00, 0.16, 0.28, 0.35, 0.10, 0.84, 0.49, 0.29, 0.06

(iii) 0.15, 0.95, 0.05, 0.88, 0.51, 0.25, 0.47

(iv) 0.60, 0.28, 0.70, 0.45, 0.46, 0.64, 0.39, 0.32, 0.57, 0.34, 0.71

Are there any instances where the Bucket Sort algorithm does not seem to be an appropriate sorting algorithm?

SOLUTION:

(i) First we instantiate a bucket $B_1, \ldots, B_{10}$ for each item in the list. On the second pass, we add each item to the appropriate bucket

Bucket	Value range	Items	Sorted
B_1	$0.00 - 0.10$		
B_2	$0.10 - 0.20$	0.15	0.15
B_3	$0.20 - 0.30$	0.26	0.26
B_4	$0.30 - 0.40$	0.33	0.33
B_5	$0.40 - 0.50$		
B_6	$0.50 - 0.60$	0.58	0.58
B_7	$0.60 - 0.70$	0.64, 0.63, 0.62	0.62, 0.63, 0.64
B_8	$0.70 - 0.80$	0.79	0.79
B_9	$0.80 - 0.90$	0.84	0.84
B_10	$0.90 - 1.00$	0.98	0.98

The only bin we needed to sort was bin B_7. We can then concatenate the sorted bins to give the sorted list:

0.15, 0.26, 0.33, 0.58, 0.62, 0.63, 0.64, 0.79, 0.84, 0.98

There are not many items that are put into the same buckets, so we do not need to do much sorting. Indeed, this list was genuinely drawn from a *uniform distribution*.

(ii) Again, we first instantiate a bucket $B_1, \ldots, B_{10}$ for each item in the list. On the second pass, we add each item to the appropriate bucket

Bucket	Value range	Items	Sorted
B_1	$0.00 - 0.10$	$0.01, 0.00, 0.06$	$0.00,\ 0.01,\ 0.06$
B_2	$0.10 - 0.20$	$0.16,\ 0.10$	$0.10\ 0.16$
B_3	$0.20 - 0.30$	$0.28,\ 0.29$	$0.28,\ 0.29$
B_4	$0.30 - 0.40$	0.35	0.35
B_5	$0.40 - 0.50$	0.49	0.49
B_6	$0.50 - 0.60$		
B_7	$0.60 - 0.70$		
B_8	$0.70 - 0.80$		
B_9	$0.80 - 0.90$	0.84	0.84
B_10	$0.90 - 1.00$		

Here we needed to sort bins B_1, B_2 and B_3 (which was already in sorted order — this could have been checked in linear time to avoid an unnecessary sorting process). We can then concatenate the sorted bins to give the sorted list:

$$0.00,\ 0.01,\ 0.06,\ 0.10,\ 0.16,\ 0.28,\ 0.29,\ 0.35,\ 0.49,\ 0.84$$

Note that there appears to be a strong preference towards the lower end of the range, resulting in more buckets that need to be sorted. Indeed, these items were not drawn from a uniform distribution (they are the squares of uniformly distributed values).

(iii) This time we instatitate only 7 buckets $B_1, \ldots, B_7$ for each item in the list. On the second pass, we add each item to the appropriate bucket

Bucket	Value range	Items	Sorted
B_1	$0.00 - 0.14$	0.05	0.05
B_2	$0.14 - 0.29$	$0.15,\ 0.25$	$0.15,\ 0.25$
B_3	$0.29 - 0.43$		
B_4	$0.43 - 0.57$	$0.51,\ 0.47$	$0.47,\ 0.51$
B_5	$0.57 - 0.71$		
B_6	$0.71 - 0.86$		
B_7	$0.86 - 1.00$	$0.95,\ 0.88$	$0.88,\ 0.95$

(ranges are rounded to 2 significant figures)

Here we needed to sort bins three buckets B_1, B_2 and B_3. We can then concatenate the sorted buckets to give the sorted list:

$$0.05,\ 0.15,\ 0.25,\ 0.47,\ 0.51,\ 0.88,\ 0.95$$

This is almost (but not quite) as bad as the case (ii) with three buckets in need of sorting. However, this distribution *is* drawn from a uniform distribution — sometimes you will get unlucky and the bucket sort algorithm will perform poorly, but these will cases will be sufficiently rare as to not affect the average-time complexity.

(iv) Finally, we have a case requiring eleven buckets $B_1, \ldots, B_{11}$. On the second pass, we add each item to the appropriate bucket

Bucket	Value range	Items	Sorted
B_1	$0.00 - 0.09$		
B_2	$0.09 - 0.18$		
B_3	$0.18 - 0.27$		
B_4	$0.27 - 0.36$	$0.28,\ 0.32,\ 0.34$	$0.28,\ 0.32,\ 0.34$
B_5	$0.36 - 0.45$	$0.45,\ 0.39$	$0.39,\ 0.45$
B_6	$0.45 - 0.55$	0.46	0.46
B_7	$0.55 - 0.64$	$0.60,\ 0.57$	$0.57,\ 0.60$
B_8	$0.64 - 0.73$	$0.70,\ 0.64\ 0.71$	$0.64,\ 0.70,\ 0.71$
B_9	$0.73 - 0.82$		
B_{10}	$0.82 - 90$		
B_{11}	$0.91 - 1.00$		

(ranges are rounded to 2 significant figures)

Here we needed to sort four bins! We can then concatenate the sorted bins to give the sorted list:

$$0.28,\ 0.32,\ 0.34,\ 0.39,\ 0.45,\ 0.46,\ 0.57,\ 0.60,\ 0.64,\ 0.70$$

This distribution was drawn from a *normal distribution* centered at around the middle of the range which explains the lack of items placed into the buckets towards the extremes of the range.

10.3 (i) Use the ideas employed in the bucket sort algorithm to construct an algorithm for sorting birthdays than has a *worst-case* time complexity of $O(n)$.

(ii) Why does this algorithm not run in linear-time for arbitrary lists of items?

SOLUTION:

(i) Since the range of possible birthdays is fixed (i.e. constant), we can adapt the Bucket Sort algorithm by first instantiating 366 Buckets for each of the days of the year. This takes at most $O(1)$ steps, even in the worst case.

We can then read each item from the list in turn and add it to the appropriate bucket without the need to compare items with each other (item comparisons are the reason for the $\Omega(n \log n)$ lower bound on general sorting algorithms). This takes at most $O(n)$ steps, since each item can be placed in the appropriate bucket in a single step.

Finally, we need to concatenate each of the buckets together. Since each of the buckets contains items with the same birthday, we do not need to sort the elements in each bucket.

A common mistake here is to suppose that since we must concatenate a fixed number (366) of list, that this can be done in constant time $O(1)$. However, list/array/string concatenation is *not* a constant time operation and takes scales linearly $\Theta(n)$ in the size of the list/array/string.

(ii) As suggested above, the reason for the efficiency of this algorithm is that it does not need to compare elements in order to sort them. This is a consequence of the fixed number of buckets.

In fact this algorithm can also be made to work in linear time in cases where we need more buckets, so long as the number of buckets is at most linear in the size of the list $O(n)$.