Homework-3

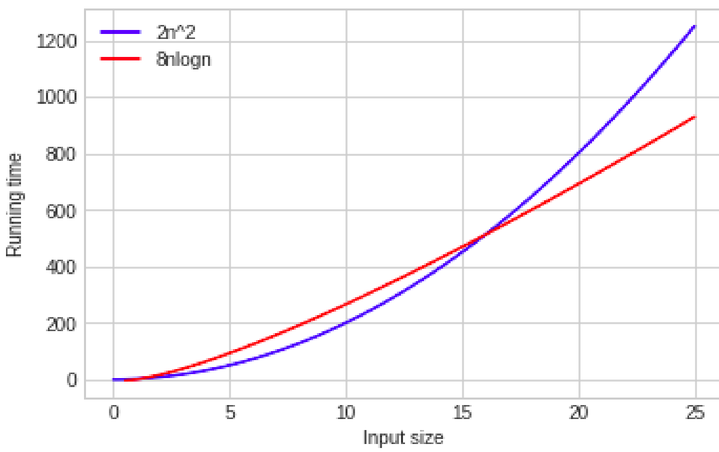
Ali Valiyev

November 2021

1 Homework solutions

Task 1

Question R-3.2: The number of operations executed by algorithms A and B are $8 * n * logn$ and $2n^2$, respectively. Determine n_0 such that A is better than B for all $n > n_0$.



Solution:

As can be seen from the graph describing the behavior of these algorithms(above) start with A higher (slower or worse) than B and eventually cross. After the point where they cross, B is always higher than A(faster or better). Therefore we need to find the point where they cross, the value where $8 * n * log(n) = 2n^2$. Applying some cancellations we get: $4 * log(n) = n$ or $4 = n/log(n)$. Solving this equation by applying logarithmic formulas, we get that n=16. But we got the point n=16 for crossing these two functions. Thus our answer is $n_0 = 17$, since for all $n \geq 17$, A will be faster than B.

Question R-3.9: Show that if d(n) is $O(f(n))$, then ad(n) is $O(f(n))$, for any constant a > 0.
Solution: Recall from the definition that: If d(n) is $O(f(n))$, then $\exists$ a constant c>0, and a constant n_0, such that $\forall$ $n \geq n_0$: d(n) $\leq$ c*f(n). Similarly, when d(n) is multiplied by a i.e when ad(n) we will have a constant ac > 0, and a constant n_0, such that $\forall$ $n \geq n_0$: ad(n) $\leq$ ac*f(n). Hence, in this time we will also have d(n) = $O(f(n))$.

Question R-3.17: Show that $(n + 1)^5$ is $O(n^5)$.
Solution: From the binomial theorem formula we know that $(n+1)^5 = n^5 + 5n^4 + 10n^2 + 5n + 1$. Clearly $n^5 + 5n^4 + 10n^2 + 5n + 1 \leq n^5 + 5n^5 + 10n^5 + 5n^5 + n^5 = 22n^5$ for $n \geq 1$. Hence k=22 and $n_0 = 1$.

Question R-3.18: Show that 2^{n+1} is $O(2^n)$.
Solution: 2^{n+1}= 2*2^n $\leq 3 * 2^n$ for $n \geq 1$. Hence k=3 and $n_0 = 1$.

Question R-3.20: Show that n^2 is $\Omega(nlogn)$.
Solution: Clearly, $nlogn \leq n * n$=n^2, since $logn \leq n$ when n$\geq$1. So, k=1 and $n_0 = 1$.

Task 2

Question R-3.15: Show that f(n) is $O(g(n))$ if and only if g(n) is $\Omega(f(n))$.
Solution: ($\Rightarrow$ part) Recall from the definition that: If f(n) is $O(g(n))$, then $\exists$ a constant c>0, and a constant n_0, such that $\forall$ $n \geq n_0$: f(n) $\leq$ c*g(n). Hence, $\exists$ a constant c>0, and a constant n_0, such that $\forall$ $n \geq n_0$: g(n) $\geq$ (1/c)*f(n).Note that: since c>0, then the constant (1/c)>0.Therefore, $\exists$ a constant k>0, namely k=(1/c), and a constant n_0, such that $\forall$ $n \geq n_0$: g(n) $\geq$ k*f(n), which is the definition of g(n) = $\Omega(f(n))$.
($\Leftarrow$ part) Recall from the definition that: If g(n) = $\Omega(f(n))$, then $\exists$ a constant k > 0, and a constant n_0, such that $\forall$ $n \geq n_0$: g(n) $\geq$ k*f(n). Hence, $\exists$ a constant k > 0, and a constant n_0, such that $\forall$ $n \geq n_0$: f(n) $\leq$ (1/k)*g(n). Note that: since k > 0, then the constant (1/k) > 0. Therefore, $\exists$ a constant c > 0, namely c = (1/k), and a constant n_0, such that $\forall$ $n \geq n_0$: f(n) $\leq$ c*g(n), which is

the definition of f(n) = $O(g(n))$.

Task 3

Question R-3.25: Give a big-Oh characterization, in terms of n, of the running time of the example3 function shown in Code Fragment 3.10.
def example3(S):
"""Return the sum of the prefix sums of sequence S."""
n = len(S)
total = 0
for j in range(n):
—for k in range(1+j):
— total += S[k]
return total
Solution: Total cost = c1 + c2 + (n+1)*c3 + n*(n+1)*c4 + n*n*c5+c6 = a*n^2+b*n+c
So, the growth-rate function for this algorithm is $O(n^2)$.

Question R-3.27: Give a big-Oh characterization, in terms of n, of the running time of the example5 function shown in Code Fragment 3.10.
def example5(A, B):
"""Return the number of elements in B equal to the sum of prefix sums in A."""
n = len(A)
count = 0
for i in range(n):
—total = 0
—for j in range(n):
— for k in range(1+j):
— total += A[k]
—if B[i] == total:
— count += 1
return count
Solution: Total cost = c1 + c2 + (n+1)*c3 + n*c4 + n*(n+1)*c5+n*n*(n+1)*c6+n*n*n*c7+n*c8+n*c9+c10 = a*n^3+b*n^2+c*n+d
So, the growth-rate function for this algorithm is $O(n^3)$.

Task 4

Question R-3.33: Solution: For a very small set of n, a $O(n^2)$ algorithm is actually faster than a $O(nlogn)$ algorithm. However, at the numbers that Al and Bob are dealing with, the most likely reason is that "Big O" notation hides a lot of details about the algorithm's runtime. Even though Al's algorithm is $O(nlogn)$, the "Big O" notation may be hiding other factors(constants) of the algorithm. Al's algorithm may have many more terms in his algorithm, but $nlogn$ is simply the biggest factor.

Task 5

Question P-3.57: Perform experimental analysis to test the hypothesis that Python's sorted method runs in O(nlogn) time on average. Solution:
import time
import numpy as np
import matplotlib.pylab as plt

def *dosorted*(length):
—bignumber = 1E2
—array = np.random.randint(0,bignumber,length)
—array = sorted(array)

pltsize = []
plttime = []
for i in np.linspace(1E3,1E5,60):
—n = int(i)
—start = time.clock()
—dosorted(n)
—end = time.clock()
—print("Sorting an array of size 0:d took 1:.6f time".format(n,(end-start)))
—pltsize.append(n)
—plttime.append(end-start)

```
plt.plot(pltsize,plttime,'o-')
plt.show()
```

The graph of complexity of a sorted function in python from size(x-axis) to time(y-axis).