Algorithms Assignment 1

Rajshree Jain

9 September 2019

1 Question 1

```
Bubble Sort(A)
1 for i=1 to A.length-1
2       for j=A.length downto i+1
3           A[j] < A [j-1]
4           exchange A[j],A[j-1]
```

a)
In order to prove that the Bubble sort actually sorts we should be able to show that that the array A' contains all the values in a sorted order but also the values in the Array A' should be comprising all the values from the array A.

b)
The loop invariant for the loop in the lines 2-4: We have can see that the inner loop iterates for every value of i such that j is a value from n down to i+1. Hence in each iteration of this inner loop we get a sub array that ranges from j to n. And the we see that after every iteration of the loop we get an array from a[j to n] where a[j] has the smallest value. When all the iterations of this inner loop are over we get an Array a[j to n] with the smallest element at a[j].

Initialization: The loop executes for values of j from n down to i+1. But in the starting there will be only one value in the loop that is A[n]. Also this will be the smallest element.

Maintenance: In this inner loop at every iteration the values of a[j] is compared with the value of a[j-1], because of this the smaller value comes at a[j-1]. At every iteration when j decreases and the smallest value keeps coming at a[j-1]. This is how actually bubble sort works. The idea of the inner loop is to bring the lowest values in the starting one by one.

Termination: When all the iterations are over the value of j will be i At this time the loop will not be executed any more. Now j=i and the sub array will be an array from a[j to n]. Further, the smallest value in a[j to n] will be placed

at a[j].

c)
Loop Invariant in the lines 1 to 4: When 1 iteration of the outer loop is over, the smallest element is placed at the position 1.
After the second iteration of the outer loop, the second smallest element is placed at position 2.
So, before the ith iteration, elements from 1 to i-1 will be the i-1 smallest elements in the n elements of the array.
After the ith iteration a[1 to i] will be the i smallest elements in the n elements of the array.

Initialization: The value of i-1 in the start will be 0, hence when the sort starts there will be no smallest element in the array.

Maintenance: We can show that the value at a[i] will be in place after any iteration and a[1 to i] will be sorted. Because for every i there will be an inner loop running that places the smallest element in the array a[j to n] at a[j-1] that is a[i]. Hence from the termination of the inner loop as seen in the upper loop invariant, we get the smallest element from a[i to n] at a[i]. This makes the array a[1 to i] sorted and a[i+1 to n] will have the remaining terms.

Termination:When the outer loop will terminate the value of i=(n), i.e. the length of the array. And at this point as seen from the maintenance the elements a[1 to n] will be present in the array in sorted order.

d) The worst case running time of Bubble Sort will be $O(n^2)$because for every i, there are approximately at max (n-i) swaps and comparisons.
The complexity of Bubble Sort will be the same in all cases i.e. best case, worst case and average case iff the above algorithm is considered. However, insertion sort has a best case sorting complexity of O(n).

2 Question 2

a)

N	1	2	3	4	5
Return value (k)	8	8	8	8	8
Return value (l)	8	384	46080	10321920	3715891200
multiplications(“*”)	6	10	14	18	22
additions(“+”)	1	1	1	1	1

when N=1
i=1

l=2.1.1=2
k=3.1.1+5=8

i=2
l=2.l.i=2.2.2=8
k=8
Multiplications=6
Additions=1

When N=2
i=1
l=2.1.1=2
k=3.1.1+5=8

i=2
l=2.2.2=8
k=8

i=3
l=2.8.3=is 48
k=8

i=4
l=2.48.4=384
k=8

Multiplications=10
Additions=1

When N=3
i=1–same as above 4 multiplications
i=2–same as above 2 multiplications
i=3–same as above 2multiplications
i=4–same as above 2 multiplications
i=5
l=2*i*1
l=2*5*384
k=8

i=6
l=2*i*1
l=2*6*(2.5.384)
k=8

l=46080

k=8
Multiplications=14
Additions=1

When N=4
the loop with i will be the same as above till 6
i=7
l=2.46080.7
k=8

i=8
l=2.645120.8
k=8

l=10321920
k=8
Multiplications=18
Additions=1

When N=5
the loop with i will be the same as above till 8
i=9
l=2.10321920.9
k=8

i=10
l=2.10321920.9.10.2=3715891200
k=8

l=3715891200
k=8
Multiplications=22
Additions=1

c)
The value of l as a function of n
We are determining l(n)
l(0)=1
l(1)=l(i=2)
l(i=1)=2.l(0).i=2.1.1
l(i=2)=2.l(i=1).i=2.2.2

l(n)=2.l(i-1).i
l(0)=2.1.1=2^1.1

$l(1) = 2.(2.1.1).2 = 2^3.1.1$
$l(2) = l(i = 4)$
$i = 3, 2.(2.2.2).3$
$i = 4, 2.(2.2.2.2.3).4$
$Hence, l(2) = 2^5.3.4$
$l(0) = 2^0.0! = 1$
$l(1) = 2^2.(2!)$
$l(2) = 2^4.(4!)$
$Hencel(n) = 2^{2n} * (2n!)$

b)When n=1 to 8,K=8, the i value that was considered in formula was 1
k=3.1.1+5
Whem n=9 to 17,K=17, the i value that was considered in formula was 2
k=3.2.2+5=17
When n=18 to 32,K=33, the i value that was considered in formula was 3
k=3.3.3+5=32
To calculate the general value of g for any particular n we must know what i value should be used for calculation
since k=3.i.i+5
i=$\sqrt{(k-5)/3}$
$i = 0.577, k = 6$
$i = 0.81, k = 7$
$i = 1, k = 8$
$i = 1.29, k = 9$
.
.
.
.
.
.
$i = 2, k = 17$
$i = 2.08, k = 18$
.
.
.
.
.
$i = 3, k = 32$
Since, i is always an integer and can never have a decimal value the corresponding values of k when i is decimal would never come in picture
so $0 <$ i $\leq 1 when\ 0 \leq n \leq 8$ but we use i=1
so $1 <$ i $\leq 2 when\ 9 \leq n \leq 17$ but we use i=2
so $2 <$ i $\leq 3 when\ 18 \leq n \leq 32$ but we use i=3
So on
Based on the above we can say that we should have an integral value of i for

any n, which can be calculated using the below ceil function as it will take care to keep the value of i integral for several values of n as required
Hence we can say that

i=ceil($\sqrt{\frac{n-5}{3}}$)

$k = 3 * ceil(\sqrt{\frac{n-5}{3}}) * ceil(\sqrt{\frac{n-5}{3}}) + 5$

d)Number of multiplications-
For calculation of l we need 2n*2 multiplications as i goes from 1 to 2n
For calculation of k we need 2 multiplications when $1 \leq n \leq 8$
For calculation of k we need 4 multiplications when $9 \leq n \leq 17$
..
and so on
Hence we can say that for k we need 2*ceil($\sqrt{\frac{n-5}{3}}$)multiplications as we go in if condition that many times
So,
Multiplications = 4n + 2*ceil($\sqrt{\frac{n-5}{3}}$)

$e) Number of Additions -$
Addition just happens in the calculation of k
We go in the if condition ceil($\sqrt{\frac{n-5}{3}}$)$times$
$Additions = ceil(\sqrt{\frac{n-5}{3}})$

3 Question 3

3.
a) $2^n + 2^n$, $\sqrt{2}^{n+20}$, 2^{2n}, 2^{n-20}, $2n^2 + 20/n$, 3^{n+30}, $nlg(n!)$
b) $(n+2)!$, $lg(n^{1.9})$, $1/n$, $lg(n)$, $n^2 lg(n^2)$, $n^{1.9} lg(n^4)$, $(n+1)!$

a) The function $2^n + 2^n$
can be written as 2.2^n

The function $\sqrt{2^{n+20}}$
can be written as $2^{n/2+10}$

The function 2^{2n}
can be written as 4^n

Comparing functions $2^{2n}, 3^{n+30}, 2^n + 2^n, \sqrt{2^{n+20}}, 2^{n-20}$

Comparing functions $4^n, 3^{n+30}, 2.2^n, 2^{n/2+10}, 2^{n-20}$

Applying limits for functions $4^n and\ \ 3^{n+30}$

$\rightarrow \lim_{n\to\infty} \frac{4^n}{3^{n+30}}$

$\rightarrow \lim_{n\to\infty} \frac{4^n}{3^{30}.3^n}$

$\rightarrow \frac{1}{3^{30}} * lim_{n\to\infty} \frac{4^n}{3^n}$

$\rightarrow \frac{1}{3^{30}} * lim_{n\to\infty} (\frac{4}{3})^n$

$\rightarrow \infty$

Hence, $2^{2n} > 3^{n+30}$

Now comparing $2^n + 2^n and\ \ 2^{n-20}$

Applying limits for functions $2.2^n and\ \ 2^{n-20}$

$\rightarrow \lim_{n\to\infty} \frac{2.2^n}{2^{n-20}}$

$\rightarrow \lim_{n\to\infty} \frac{2.2^n}{2^n.2^{-20}}$

$\rightarrow \frac{2^{21}}{1} * lim_{n\to\infty} \frac{2^n}{2^n}$

$\rightarrow \frac{2^{21}}{1}$

Hence, the order of $2^n + 2^n and\ \ 2^{n-20}\ \ is\ \ same$

Now comparing $2^n + 2^n and\ \ 3^{n+30}$

Applying limits for functions $2.2^n and\ \ 3^{n+30}$

$\rightarrow \lim_{n\to\infty} \frac{2.2^n}{3^{n+30}}$

$\rightarrow \lim_{n\to\infty} \frac{2.2^n}{3^n.3^{30}}$

$\rightarrow \frac{2}{3^{30}} * lim_{n\to\infty} \frac{2^n}{3^n}$

$\rightarrow 0$

Hence $3^{3+30} > 2^n + 2^n$

The order till now is $2^{2n}, 3^{n+30}, (2.2^n)^*, (2^{n-20})^*$

Applying limits for functions $2^{n-20} and\ \ 2^{n/2+10}$

$\rightarrow \lim_{n\to\infty} \frac{2^{n-20}}{2^{n/2+10}}$

$\rightarrow \lim_{n\to\infty} \frac{2^{n}}{2^{30}.2^{n/2}}$

$\rightarrow \frac{1}{2^{30}} * lim_{n\to\infty} \frac{2^{n}}{2^{n/2}}$

$\rightarrow \frac{1}{2^{30}} * lim_{n\to\infty} (\frac{2}{1})^{n/2}$

$\rightarrow \infty$

Hence $2^{n-20} > \sqrt{2^{n+20}}$

Order now will be $2^{2n}, 3^{n+30}, (2.2^{n})^{*}, 2^{n-20}, \sqrt{2^{n+20}}$

The functions that are left are nlg(n!) and $2\mathrm{n}^2 + 20/n$

n! can be written as n^n

So applying limits to nlg(n^n) *and* $2n^2 + 20/n$

$\rightarrow \lim_{n\to\infty} \frac{nlg(n^{n})}{2n^{2}+20/n}$

$\rightarrow \lim_{n\to\infty} \frac{n^{2}.lg(n)}{2.n^{2}+20/n^{3}}$

Applying L' Hopitals rule and differentiating the Numerator and Denominator

$\rightarrow lim_{n\to\infty} \frac{lgn}{2+20/n^{5}}$

$\rightarrow lim_{n\to\infty} \frac{1/n}{-100/n^{6}}$

$\rightarrow lim_{n\to\infty} \frac{n^{5}}{-100}$

$\rightarrow \infty$

hence, n.lg(n!)$> 2.\mathrm{n}^2 + 20/n$

Now comparing n.lg(n!) and $2^{n/2+10}$

Applying limits for functions n.lg(n!) and $2^{n/2+10}$

$\rightarrow \lim_{n\to\infty} \frac{n.lg(n!)}{2^{n/2+10}}$

$\rightarrow \lim_{n\to\infty} \frac{n^2 lg(n)}{2^{n/2+10}}$

Applying L Hopital's rule and differentiating the numerator and denominator

$\rightarrow \lim_{n\to\infty} \frac{2nlg(n)+n}{\frac{2^{20}.2^{n/2}lg2}{2}}$

$\rightarrow \frac{1}{2^{20}} * \lim_{n\to\infty} \frac{4nlg(n)+2n}{2^{n/2}lg2}$

$\rightarrow \frac{1}{2^{18}*lg2*lg2} * \lim_{n\to\infty} \frac{2lg(n)+3}{2^{n/2}}$

$\rightarrow \frac{1}{2^{18}*lg2*lg2} * \lim_{n\to\infty} \frac{2lg(n)+3}{2^{n/2}}$
$\rightarrow \frac{1}{2^{19}*lg2*lg2} * \lim_{n\to\infty} \frac{2/n}{2^{n/2}lg2}$
$\rightarrow \frac{1}{2^{18}*lg2*lg2} * \lim_{n\to\infty} \frac{2}{n.2^{n/2}}$
$\rightarrow 0$
Hence, n.lg(n!) $< 2^{n/2+10}$

The order will be finally: $2^{2n}, 3^{n+30}, (2.2^n)^*, 2^{n-20}*, (\sqrt{2^{n+20}})^*, nlog(n!), 2n^2 + 20/n$

b) We will start with comparing the functions $\lg(n^{1.9}), 1/n, lg(n), n^2lg(n^2), n^{1.9}lg(n^4)$

We will start with $\lg(n^{1.9}) and 1/n$
$\rightarrow \lim_{n\to\infty} \frac{lg(n^{1.9})}{1/n}$
$\rightarrow \lim_{n\to\infty} \frac{1.9*lg(n)}{1/n}$
$\rightarrow \lim_{n\to\infty} 1.9 * n * lg(n)$
$\rightarrow \infty$
Hence $\lg(n^{1.9}) > 1/n$

Then we compare $\lg(n^{1.9})$ *and* $lg(n)$
$\rightarrow \lim_{n\to\infty} \frac{lg(n^{1.9})}{lg(n)}$
$\rightarrow \lim_{n\to\infty} \frac{1.9*lg(n)}{lg(n)}$
$\rightarrow 1.9$
i.e. greater than 0 and hence they are asymptotically same

so lg(n1.9)*,lg(n)*,1/n

Now, we compare $n^2lg(n^2)$ *with* $lg(n)$
$\rightarrow \lim_{n\to\infty} \frac{n^2lg(n^2)}{lg(n)}$
$\rightarrow \lim_{n\to\infty} \frac{2*n^2lg(n)}{lg(n)}$

$\rightarrow \lim_{n\to\infty} \frac{2*n^2}{1}$
$\rightarrow \infty$

so $n^2 lg(n^2), lg(n^{1.9})^*, lg(n)^*, 1/n$

Now we compare $n^2 lg(n^2)$ *and* $n^{1.9} lg(n^4)$
$\rightarrow \lim_{n\to\infty} \frac{n^2 lg(n^2)}{n^{1.9} lg(n^4)}$
$\rightarrow \lim_{n\to\infty} \frac{2*n^2 lg(n)}{4*n^{1.9} lg(n)}$
$\rightarrow \lim_{n\to\infty} \frac{1*n^2}{2*n^{1.9}}$
$\rightarrow \infty$

so $n^2 lg(n^2) > n^{1.9} lg(n^4)$

so now we will compare $n^{1.9} lg(n^4)$ *and* $lg(n^{1.9})$
$\rightarrow \lim_{n\to\infty} \frac{n^{1.9} lg(n^4)}{lg(n^{1.9})}$
$\rightarrow \lim_{n\to\infty} \frac{4*n^{1.9} lg(n)}{1.9*lg(n)}$
$\rightarrow \infty$

so $n^2 lg(n^2), n^{1.9} lg(n^4), lg(n^{1.9})^*, lg(n)^*, 1/n$

Now comparing (n+1)! and $n^2 lg(n^2)$
$\rightarrow (n+1)! = \omega(2^{n+1})$

$\rightarrow Comparing(2^{n+1}) and\ n^2 lg(n^2)$

$\rightarrow \lim_{n\to\infty} \frac{n^2 lg(n^2)}{2^{n+1}}$
$\rightarrow \lim_{n\to\infty} \frac{2.n^2 lg(n)}{2^{n+1}}$
$\rightarrow \lim_{n\to\infty} \frac{n^2 lg(n)}{2^n}$

Applying L Hopitals rule and limit n tending to ∞
$\rightarrow \lim_{n\to\infty} \frac{2.lg(n)+n}{2^n lg(2)}$

Applying L Hopitals rule and limit n tending to $\infty again$
$\rightarrow \lim_{n\to\infty} \frac{2.lg(n)+2+1}{2^n lg(2) lg(n)}$

Applying L Hopitals rule and limit n tending to $\infty again$
$\rightarrow \lim_{n\to\infty} \frac{2/n}{2^n lg(2) lg(n)}$

Applying limits we are getting 0

$\rightarrow (2^{n+1}) = \omega(n^2.lg(n^2))$
$\rightarrow$
(n+1)!=$\omega(2^{n+1})$

→

(n+1)!=$\omega(n^2.lg(n^2))$ $---->$ *transitivity*

so (n+1)!,$n^2lg(n^2), n^{1.9}lg(n^4), lg(n^{1.9})^*, lg(n)^*, 1/n$

Now we compare (n+2)! and (n+1)! in the end

→$\lim_{n\to\infty} \frac{(n+2)!}{(n+1)!}$
→$\lim_{n\to\infty} \frac{(n+2)(n+1)!}{(n+1)!}$

→$\lim_{n\to\infty}(n+2)$

$\to \infty$

So we can say that (n+2)!>(n+1)!

The final order is (n+2)!,(n+1)!,$n^2lg(n^2), n^{1.9}lg(n^4), lg(n^{1.9})^*, lg(n)^*, 1/n$

4 Question 4

a) lg(n)+n ϵ $o(n)(little - o)$

When we say that f(n) is "little o" of g(n) i.e. f(n)$\epsilon o(g(n))$it means that:
$0 \leq f(n) < c * g(n), \forall n \geq n_0(c)$,for every c >0
→ $0 \leq nlg(n) + n < c * n$ $---->$to prove
→ $0 \leq nlg(n) + n$, this will be true $\forall n > 1$
→ $nlg(n) + n < c * n$
→ $lg(n) + 1 < c$
→ $lg(n) + 1 < c$ $---->$ Here the value of lg(n) keeps growing with the value of n but c on the RHS is constant.
Hence lg(n)+n $\notin$ $o(n)$

b)$3n^2 - sin(n) - n$ ϵ $\theta(n^2)$
When we say that f(n)=$\theta g(n)$
there exist positive constants c1, c2, and n_0 such that $0 \leq c1 * g(n) \leq f(n) \leq c_2 * g(n)$ for all n $\geq n0$
that means that for "big-theta" f(n) ϵ $O(g(n))and$ $f(n)\epsilon$ $\Omega(g(n))$
→ $0 \leq 3n^2 - sin(n) - n \leq c * n^2$
→ $0 \leq 3n^2 - sin(n) - n$ $\forall n \geq 0$

→ $3n^2 - sin(n) - n \leq c * n^2$
→ $3n^2 - sin(n) - n \leq 3n^2$

$\rightarrow 3n^2 - sin(n) - n \leq 3n^2 + n^2 + n^2$ as we know that -sin(n) $\leq n^2$ $and - n \leq n^2$ $\forall n \geq 0$
$\rightarrow 3n^2 - sin(n) - n \leq 5n^2$
for c=5 and n ≥ 0 $3n^2 - sin(n) - n = O(n^2)$

Now we will show that $3n^2 - sin(n) - n = \Omega(n^2)$
$\rightarrow 3n^2 - sin(n) \geq c * (n^2) \geq 0$
$\rightarrow c * (n^2) \geq 0 \ \forall n \geq 0$
$\rightarrow c * (n^2) \leq 3n^2 - sin(n) - n$
$\rightarrow 3n^2 - sin(n) - n \leq 3n^2 - sin(n) - n$
$\rightarrow 3n^2 - n^2 - n^2 \leq 3n^2 - sin(n) - n$
$\rightarrow n^2 \leq 3n^2 - sin(n) - n \ \forall n \geq 0$
Hence $3n^2 - sin(n) - n = \Omega(n^2)$

Since $3n^2 - sin(n) - n = O(n^2)$ and $3n^2 - sin(n) - n = \Omega(n^2)$

$Hence, 3n^2 - sin(n) - n \ \epsilon \ \theta(n^2)$

$c) \sum_{i=1}^{3n} i \in O\sqrt{n}$

$\rightarrow 0 \leq \frac{(3n)(3n+1)}{2} \leq c * n^{1/2}$
$\rightarrow 0 \leq \frac{(3n)(3n+1)}{2} \ \forall n \geq 1$
$\rightarrow 4.5.n^2 + 1.5 * n \ \leq c.n^{1/2} \ \forall n \geq 1$
$\rightarrow 4.5.n^{3/2} + 1.5 * n^{1/2} \ \leq c \ \forall n \geq 1$
→But this is not possible as LHS will grow with the value of n and RHS is fix
→Hence $\sum_{i=1}^{3n} i \notin O(n^2)$

$d) \ nlg(n) + n^2 \in \omega(n)$

→ If f(n)=$\omega(g(n))$ for any positive constant c > 0, there exists a constant $n_0 > 0$ $such \ that \ 0 \leq c.g(n) < f(n) \ \forall n > n_0$
$\rightarrow nlg(n) + n^2 > c * n$
$\rightarrow lg(n) + n > c \ \forall n \geq 1$
→ Here the RHS grows with the value n and LHS is bound
$\rightarrow Hence, \ nlg(n) + n^2 \in \omega(n)$

5 Question 5

A non recursive algorithm that calculates the value of $(3a)^{n/2}$where we will be given the value of a and n in the time complexity of $\Theta(lg(n))$
a is positive
n is positive but not a power of 2

A textual description of the Algorithm

1. We have to calculate the $3a^{n/2}$
2. n can be even or odd, depending on which the value of n/2 can be an integer or a fraction with denominator 2
3. If the value of n/2 is an integer (all times when n is even) we will just calculate the final value using a loop
4. If the value of n/2 is a fraction (in case when n is odd) we will write the fraction in form (n-1)/2+1/2
5. Looking at point 4. we can say that when ever the value of n is odd we need to have a square root of the base calculated i.e $\sqrt{3a}$
6. So we will keep the square root calculated and the rest left over integral power we can calculate in a loop
7. Now how to calculate the integral power using a loop
8. If the power is even and in form of 2^x for eg. power=8 then it will be easy to iterate the loop and calculate the power by just multiplying the base with itself when ever we divide the power by 2
9.But when the power is not in form of 2^x eg. power=5 we need to have some extra powers multiplied in the answer like $base^4 * base^1$
10. This we will deal in the loop by checking that when ever the value of power is odd we collect the extra terms to be multiplied using an if condition that multiplies the previous base with the current value of FinalAnswer.
11. When in the end the value of power reduces to 1, the value of FinalAnswer and Answer is multiplied to club the entire result
12. Finally we check if the value of check=1 that means the actual final power is in form 4.5, 5.5, 6.5 etc. We multiply the FinalAnswer calculated by the loop with the root value.

Algorithm

```
1 a=value
2 n=value
3 FinalAnswer=1
4 Answer=(3*a)
5 root=sqrt(3*a)
6 if n is multiple of 2
7        check=0
8 else
9       check=1
10 power=(n/2)
11 while power ≥ 1
12          if power is not multiple of 2
13                FinalAnswer=FinalAnswer*(Answer)
14          Answer=Answer*Answer
```

15 power=(power/2)
16 $if check == 1$
17 $FinalAnswer = FinalAnswer * root$
18 $return(FinalAnswer)$

Time Complexity

T(n)= Time for the while loop to run + constant time for other statements
Time taken for the while loop to run -
n/2 $\rightarrow n/4 \rightarrow n/8 \rightarrow -------n/2^k$

The termination condition for the loop will be when $n/2^k = 1$
In that case $n = 2^k$
lg(n) = k
Hence the loop is going to run k times
The 2 statements in the loop will take constant time
So, k.c is the time taken in the loop
That is lg(n).c
The time taken by the other statements in the code will also be constant
Hence lg(n).c+c1

T(n)=O(lg(n))
T(n) $\leq lg(n)$
$We can also write T(n) \geq lg(n)$
$Hence, T(n) = \Theta(lg(n))$

Explain with Example

Let us say the value of n=9
Let us say a=3
Answer = 3*3 = 9
FinalAnswer=1
root = 3
n is not a multiple of 2 hence, check=1
power=(9/2) = 4

Iteration 1:
power ≥ 1
$power is even hence not goes in if condition$
$Answer = (3.3).(3.3)$
$power = 4/2 = 2$

$Iteration 2:$

$power \geq 1$
$Answer = (9).(9).(9).(9)$
$power = 2/2 = 1$

$Iteration 3 : Power \geq 1$
Since power is odd the value of FinalAnswer=FinalAnswer*Answer=$(9)^4$

$check = 1, hence Answer = (9).(9).(9).(9).(3)$

$- -$
$- -$

Let us take another example where n = 10
Let us a = 3
Answer = 3*3 = 9
root = 3
FinalAnswer=1
n is a multiple of 2
check = 0
power = 5

Iteration 1:
FinalAnswer=FinalAnswer.(9) = 9 since power was odd
Answer=(9)(9)
power=5/2=2

Iteration 2:
Answer=9.9.9.9
power=2.5/2=1

Iteration 3:
FinalAnswer=FinalAnswer*Answer=9.$(9)^4$
$Answer = (9)^8$
$power = 1/2 = 0$

Loop ends and the FinalAnswer 9^5 is returned

Proof of correctness of the Algorithm

Case 1. The value of power is in form 2^k

$The Algorithm works correctly as it iterates k times and each time does :$
$base = base * base$
$Hence, the answer is correct.$

$Case 2. The value of power is not in the form of 2^k$
Case 2a. The value of power is even
Case 2b. The value of power is odd
In this case the value of base is multiplied by the final answer when ever the value of power is odd

Since, for all types of numbers the answer is correctly determined by the algorithm and it terminates in lg(n) time, the algorithm is correct