1.

i.

Since rand() always returns a value between 0 and 1, so the best case is rand() = 1
The worst case is rand() < 0.5; and average case would be between 0 and 1.

a. Best case: 1(initialize) +6(checks) +1(increasement) + 2(increasement) = 10 operations
   Worst case: 1(initialize) +6(checks)+ 1(increasement) +6(increasement) = 14 operations
   Average case: 1(initialize) +1(increasement) +4(increasement) = 12 operations
b. Best case, worst case, average case: Ω (1), O(1), Θ(1)
c. f = O(1)
d. f = Ω(1)
e. f = Θ(1)

ii.

Best case would be rand() is >=0.5, worst and average case would be N>0 with rand() < 0.5

a. Best case: 1 + 1 + n +1 + n + n + n = 4n+3 operations
   Worst case: 1 + 1 + (n + 1) + n + n + n + n = 5n+3 operations
   Average case: (9n)/2 +3
b. Best case: Ω(n)
   Worst case: O(n)
   Average case: Θ(n)
c. f = O(n)
d. f = Ω(n)
e. f = Θ(n) //sorry for typo

iii.

Best case would be unlucky is always false
Worst case would be unlucky == true

a. Best case: 1+1+1 + n + n +n = 3n+3 operations
   Worst case: 1(Initial "count") +1(initial "i") +n+1(checks N) + n(check unlucky)+n(assign "j") + ((n+1)/2 +1)(checks) + n(n+1)/2(increasement) + n(n+1)/2(decrement) +n (increment) = 3n^2/2 + 13n/2 + 3 operations
   Average case: 3n^2/4 + 19n/4 + 3
b. Best case: Ω(n)
   Worst case: O(n^2)
   Average case: Θ(n^2)
c. f = O(n^2)
d. f = Ω (1)

iv.

Best case would be unlucky == false so the loop doesn't run
Worst case would be lucky == true and N>0 so the loop can run

a. Best case: 1(Init "count) + 1(Init "I") + 1 (check unlucky) = 3 operations
   Worst case: 1(Init "count") + 1 (Init "I") + 1 (check unlucky) + log(n)(check "i") + log(n)(increment for "count") + log(n)(decrement for "i") = 3log(n)+3 operations
   Average case: 3log(n)/2 + 3 operations
b. Best case: Ω(1)
   Worst case: O(log(n))

Average case: O(log(n))

c. f = O(log(n))
d. f = Ω(1)

v.
The best case would be rand()>=0.5 so that the count remains 0, the second loop also inactive
The worst case would be N> 0 then the 2 loops can run, rand() < 0.5

a. Best case: 1(init “count”) + 1(init “i”) + n+1(check “i”) + n (init “num”) +(check “num”) + n(i++) + 1(init “num”) + 1 (init “j”) + 1 (check “j”) = 3n + 6 operations
   Worst case: 1(init “count”) + 1(init “i”) + n (check “i”) + n (init “num”) + n(check “num”) + n(“count” increment) + n(“i” increment) + 1(init “num”) +1 (init “j”) + n(check “j”) + n(“count” increment) + n(“j” increment) = 8n +4 operations
   Average case: 11n/2+5 operations
b. Best case : Ω(n)
   Worst case: O(n)
   Average case: Θ(n)
c. f = O(n)
d. f = Ω(n)
e. f = Θ(n)

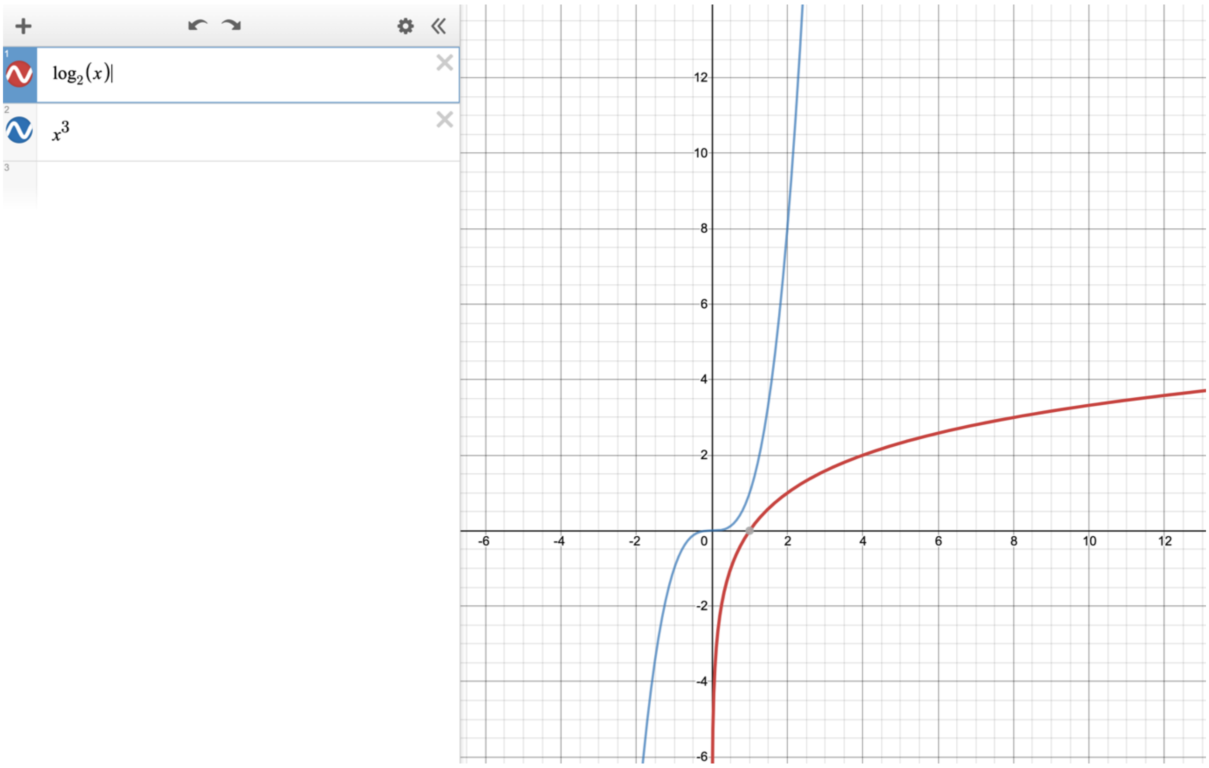
vi.
the best case would be a[j] > a[j+1] so the loops doesn’t run
the worst case would be N> 1

a. Best case: 1(init “i”) + n-1(check) +n-1(init “j”) + n(n+1)/2-2 (check) + n(n+1)/2-1 (check) + n(n+1)/2-2(increment) n-1(increment) = 3x^2/2 + 9x/2 -7
   Worst case: 1(init “i”) + n-1(check) +n-1(init “j”) + n(n+1)/2-2 (check) + n(n+1)/2-1 (check) + n(n+1)/2-2 (swap) + n(n+1)/2-2(increment) n-1(increment) = 2x^2 + 5x -10 operations
   Average: (7x^2 +19x)/4 -17/2
b. Best case : Ω(n^2)
   Worst case: O(n^2)
   Average case: Θ(n^2)
c. f= O(n^2)
d. f = Ω(n^2)
e. f = Θ(n^2)

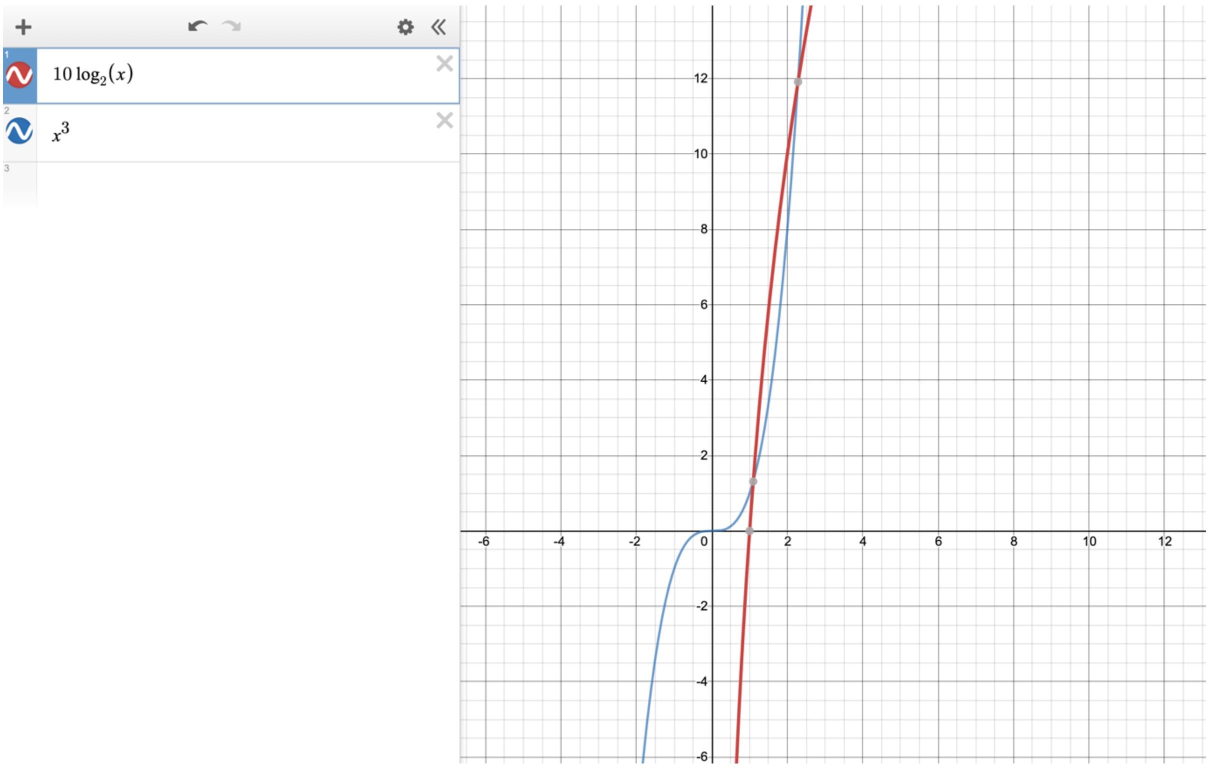
f. The best way to describe the performance of algorithms is using big O notation, because it represent the worst case scenarios, which are regularly occurs during normal usage.

2.
Big O describe the worst-case scenario, can be used to describe the execution time or space used by algorithms.

3. Θ(n^3) not always takes longer to run than Θ(log(n)). Because it depends on how algorithms process the operations. For example:



Comparison between log(x) and x^3 algorithm



Comparison between 10log(x) and x^3

4.

True, the highest power of variable n is 2, therefore O(n^2)

False, because

$$\lim_{n \to \infty} \frac{n}{n \log{(n)}} = \lim_{n \to \infty} \frac{1}{n} = 0$$

Hence nlog(n) grows faster than n,
so O(n) can not be the upper bound of nlog(n)

False, as n^4 is not a lower bound to n^3
True, because n log(n) grows faster than n, so n is a lower bound for it