Q1

$$\lim_{n\to\infty} \frac{T_2(n)}{T_1(n)} = \lim_{n\to\infty} \frac{4\log(\log n)}{3\log n+3} = \lim_{n\to\infty} \frac{4\ln(\ln n)}{3\ln n+3}$$

$$= \lim_{n\to\infty} \frac{4\frac{1}{n\ln n}}{3\frac{1}{n}} = \lim_{n\to\infty} \frac{4}{3\ln n} = 0$$

$\Rightarrow T_2(n) \in O(T_1(n))$

$$\lim_{n\to\infty} \frac{T_1(n)}{T_4(n)} = \lim_{n\to\infty} \frac{3\log n+3}{2000n+1} = \lim_{n\to\infty} \frac{3\ln n+3}{2000n+1}$$

$$= \lim_{n\to\infty} \frac{3\frac{1}{n}}{2000} = \lim_{n\to\infty} \frac{3}{2000n} = 0$$

$\Rightarrow T_1(n) \in O(T_4(n))$

$$\lim_{n\to\infty} \frac{T_4(n)}{T_5(n)} = \lim_{n\to\infty} \frac{2000n+1}{n^2/36} = \lim_{n\to\infty} \frac{2000}{n/18} = 0$$

$\Rightarrow T_4(n) \in O(T_5(n))$

$$\lim_{n\to\infty} \frac{T_5(n)}{T_3(n)} = \lim_{n\to\infty} \frac{n^2/36}{n^5+8n^4} = \lim_{n\to\infty} \frac{1}{36(n^3+8n^2)} = 0$$

$\Rightarrow T_5(n) \in O(T_3(n))$

$$\lim_{n\to\infty} \frac{T_3(n)}{T_8(n)} = \lim_{n\to\infty} \frac{n^5+8n^4}{2^n+n^3} = \lim_{n\to\infty} \frac{5n^4+32n^3}{2^n\ln 2+3n^2}$$

$$= \lim_{n\to\infty} \frac{1}{2^n(\ln 2)^6} = 0$$

$\Rightarrow T_3(n) \in T_8(n)$

$$\lim_{n\to\infty}\frac{J_8(n)}{J_6(n)} = \lim_{n\to\infty}\frac{2^n+n^3}{3^n+n^4} = \lim_{n\to\infty}\frac{2^n(\ln 2)^4}{3^n(\ln 3)^4}$$

$$= \lim_{n\to\infty}\left(\frac{2}{3}\right)^n\cdot\left(\frac{\ln 2}{\ln 3}\right)^4 = 0 \Rightarrow J_8(n)\in O(J_6(n))$$

$$\lim_{n\to\infty}\frac{J_6(n)}{J_7(n)} = \lim_{n\to\infty}\frac{3^n+n^2}{n^7+1000n} = \lim_{n\to\infty}\frac{\frac{3^n}{n^6}+1/n}{n^{n-3}+\frac{1000}{n^2}}$$

$$= \lim_{n\to\infty}\frac{3^n/n^3}{n^{n-3}} = \lim_{n\to\infty}\left(\frac{3}{n}\right)^n = \lim_{n\to\infty} 0^n = 0$$

$$\Rightarrow J_6(n)\in O(J_7(n))$$

Therefore sorted order (answer to Q1) is:

J_2, J_1, J_4, J_5, J_3, J_8, J_6, J_7 — — —

Q2

a) $$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{99n}{n} = 99 \Rightarrow f(n)\in\Theta(g(n))$$

b) $$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{2n^4+n^2}{(\log n)^6} = \lim_{n\to\infty}\frac{\sqrt[6]{2n^4+n^2}}{\ln n}$$

$$= \lim_{n\to\infty} n\cdot\frac{4n^3+n}{3(2n^4+n^2)^{5/6}} = \lim_{n\to\infty}\frac{4n^4+n^2}{3(2n^4+n^2)^{5/6}}$$

$$= \lim_{n\to\infty}\frac{4}{3}\cdot\frac{n^4+n^2}{2^{5/6}\cdot n^{4\cdot\frac{5}{6}}+\cdots} = \lim_{n\to\infty}\frac{4}{3\cdot 2^{5/6}}\cdot\frac{n^4+n^2}{n^{4\cdot\frac{5}{6}}+\cdots}$$

$$= \lim_{n\to\infty}\frac{4}{3\cdot 2^{5/6}}\cdot\left(n^{\frac{2}{3}}+\cdots\right) = \infty \Rightarrow f(n)\in\Omega(g(n))$$

Q2

c) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\sum_{k=1}^{n} k}{4n+\log n} = \lim_{n\to\infty} \frac{\frac{n(n+1)}{2}}{4n+\ln n}$

$= \lim_{n\to\infty} \frac{n^2+n}{8n+2\ln n} = \lim_{n\to\infty} \frac{2n+1}{8+\frac{2}{n}} = \lim_{n\to\infty} \frac{2n+1}{8} = \infty$

$\Rightarrow f(n) \in \Omega(g(n))$

d) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3^n}{5^{\sqrt{n}}} = \lim_{n\to\infty} \frac{n \ln 3}{\sqrt{n} \ln 5}$

$= \lim_{n\to\infty} \sqrt{n} \cdot \frac{\ln 3}{\ln 5} = \infty \Rightarrow f(n) \in \Omega(g(n))$

Q3

a) The algorithm traverses the given array and counts how many times each item occurs without the items prior to it.

input "nums" is the array and input "n" is the array length.

output is the first item in the array that occurs n/2 times in the array after itself or -1 if no such item exists.

b) Best case is when first item in the array is the first item that occurs n/2 times. It still needs to count it n/2 times therefore its $\Theta(n/2) = \Theta(n)$.

Worst case is when no such item is found or the item in the middle is the satisfying item

If such item is not found for every item in the array their leading items are checked therefore it is $\Theta((n-1)(n)/2)$ which equals to $\Theta(n^2)$

If the middle element is the satisfying item then first half of the array is not therefore it is

$$\Theta\left(\frac{(n-1)n}{2} - \frac{(n/2)(n/2-1)}{2}\right)$$

$$= \Theta\left(\frac{n^2-n}{2} - \frac{n^2/4 - n/2}{2}\right)$$

$$= \Theta\left(\frac{3n^2/4 - n/2}{2}\right) = \Theta(n^2)$$

Q4/

a) the algorithm finds the biggest number in the array and allocates another array with the size of this biggest number.

Then the given array is traversed again and for each item their relative array value is incremented for each occurence.

Then the array containing occurence numbers is traversed to get the satisfying number (which occurs more than n/2 times) if there's any.

input "nums" is the array and input "n" is the array length.

output is the first item to occur more than n/2 times in the array. if there's none then it's -1.

b) The best case and worst case are equal since algorithm traverses the whole array to find max element in any case and the loops after it are linear complexity at max.

Therefore both are $\Theta(n)$.

Still best case is faster than worst case with having first element as the satisfying item but the difference is third of worst performance.

Q5| Let $A3$ be the algorithm in Q3 and $A4$ be the one in Q4.

$A3$ only uses one integer other than its arguments for space considerations making it constant sized.

However it is considerably slow. Considering the content of given array doesn't necessarily has a structure, making it random, there's little to no chance of it having a satisfying item therefore being $\Theta(n^2)$.

With same thought process it can be seen that size of map array of $A4$ is not necessarily dependent on the size of the array and making it susceptible to highly inefficient space sizes. For $A4$ space can not be even calculated meaningfully.

For time complexity $A4$ is strictly faster at $\Theta(n)$

Q6)

a)

```
int max_axb(int A[], int B[], int n, int m) {

    int a_max = A[0];
    for(int i=1; i<n; i++) {
        if(A[i] > a_max) a_max = A[i];
    }
    int b_max = B[0];
    for(int i=1; i<m; i++) {
        if(B[i] > b_max) b_max = B[i];
    }
    return a_max * b_max;
}
```

aa) the best and worst cases of algorithm are the same since there is only one return keyword and no other jump or break keywords.

Both arrays are fully traversed to find max elements of each therefore the algorithm's time complexity is $\Theta(n+m)$

b)

```
func sort_merged(A, B)
    I ← merge A B
    l ← length I
    for i from 0 to l-2
        for j from i to l-2
            if I[j] < I[j+1] then
                swap I[i] I[j+1]
            end if
        end for
    end for
    return I
end func
```

b. a)

The best and worst cases of algorithm are equal since there's no return or break etc. keywords other than the one at last line.

The merge operation is $\Theta(n+m)$ which creates an array of length $n+m$ and copies elements of A and B.

Then two nested for loops are iterated with $l = n+m$. The swap operation is $\Theta(1)$ and number of comparisons done is

$$(l-1)+(l-2)+\cdots+2+1 = \frac{(l-1)\,l}{2} = \frac{l^2-l}{2}$$

and since $\frac{l^2-l}{2} \in \Theta(l^2)$ the function is quadratic

c)

```
func add_item(A   x)
  l ← length A
  C ← array l+1
  for i from 0 to l-1
    C[i] ← A[i]
  end for
  C[l] ← x
  return C
end func
```

c.a)

Again the function has equal best and worst case time complexities due to having only one control-flow keyword.

The allocation and length operations are $\Theta(1)$. The number of assignments of elements of new array C is $l+1$ therefore function's time complexity is $\Theta(l+1) = \Theta(l)$ or if given array A is length of n then $\Theta(n)$ therefore linear.

d)

```
func del_item (A   x)
  l ← length A
  C ← array l-1
  j ← 0
  for i from 0 to l-1
    if A[i] ≠ x then
      C[j] ← A[i]
      j ← j+1
    end if
  end for
  return C
end func
```

d. a)

The function's best and worst case time complexities are same due to only one control-flow instruction.

The time complexity of the function is linear since the comparison is done for every element of given array exactly once. And the other operations such as array allocation are constant time.