1. Graded by Manna

\[ T(n) = T(\sqrt{n}) + 1 = T(n^{\frac{1}{2}}) + 1 = T(n^{\frac{1}{2^2}}) + 2 = \ldots = T(n^{\frac{1}{2^i}}) + i. \]

Assume \( n^{\frac{1}{2^i}} = c \) is some small constant such that \( T(c) = \) constant. Thus, \( T(n) = T(c^i) + i = T(c) + i = O(i) \). Now, since \( c^i = n \), \( \log n = 2^i \log c \), which implies \( i = \log \log n - \log \log c = O(\log \log n) \). Therefore, \( T(n) = O(\log \log n) \).

Many of you gave answers like \( O(\log n) \) and ended up getting max 30-40% marks. The reason penalties are harsh is because this problem was one where you could get the answer by simply computing mechanically – no special intuition needed to be shown. In such cases, in order to get full credit, you must get it perfect. Moral of the story: *easy come, easy go.*

2. Graded by Ravi

We get an \( O(1) \) amortized cost for Extract-min by using leftover credit from Insert. An insert of an element is charged \( O(\log n) \), the depth of the heap, but the actual cost is only the number of levels the element has to percolate up. The rest of the credit (equal to the depth of the node at which the element is positioned) is used for achieving a \( O(1) \) amortized cost for Extract-min.

We define a potential function based on the depth of nodes in the heap. Depth is the number of edges need to traversed to reach a root from node.

Let \( H_i \) be the heap after \( i \)th operation. For a node \( x \) let \( d_i(x) \) be the depth of \( x \) in \( H_i \).

\[
\Phi(H_i) = \sum_{x \in H_i} c \left( d_i(x) + 1 \right)
\]

Where \( c \) is a constant.

Initially heap is empty, so \( \Phi(D_0) \) is 0. So the potential function is always positive.

We know that

\[
C_{am} = C_{ac} + \Delta \Phi
\]
For an insert the potential changes by the depth of new node inserted, which is $c(1 + \text{floor}(\log n_i))$. The actual cost of insert is $O(\log n_i)$. So the amortized cost is $O(\log n)$.

For extract-min, the potential decreased by depth of the last node in the heap. This is so because in an extract-min root node is removed and the last node is percolated down from the root. So overall the heap structure changes by only the last node. So change is potential is $1 - c(1 + \text{floor}(\log n_{i-1}))$

Amortized cost is

$$c \log n_{i-1} - c(1 + \text{floor}(\log n_{i-1}))$$

Which is $O(1)$.

Answers explaining the idea without giving actual equations got 18-22 points. If you give potential function correct but not the amortized cost (or vice versa, based on good arguments), you receive 10-16 points.

3. Graded by Ravi

For this problem we first find non-subsumed tuples and then remove them from total tuples.

This can be solved by sorting the tuples on $a$th dimension (in descending order) and iterating over that dimension from largest to smallest value. During the $i$th iteration ($i$ from 1 to $n$) we maintain the max-tuple, which is the tuple with maximum value of $b$ in 1 to $i$. Any tuple in $i + 1$st location is subsumed by any tuple in the list if and only if it is subsumed by this max-tuple.


Finding the non-subsumed tuples:

First sort the array $A$ on $a$ in descending order, call the sorted array $A_s$. So we have the largest value of $a$’s at $A_s[0].a$.

From the definition of subsumption, $A_s[0]$ can not be subsumed since there is no $A_s[i].a$ greater than $A_s[0].a$.

Now, we know that all $A_s[i]$, $i \geq 1$ such that $A_s[i].b \leq A_s[0].b$ are subsumed.

Our algorithm generalizes the above idea. We iterate on $A_s$ from 1 to $n$. At iteration $i$ we remember a tuple $A_s[max]$ such that $A_s[max].b$ is greater than all $A_s[j].b$ for $i \leq j \leq n$.

Now if $A_s[i + 1].b \leq A_s[max].b$ then $A_s[i + 1]$ is subsumed. Otherwise update $max = i + 1$ and put tuple $A_s[i + 1]$ is tuples which are not subsumed.

At the end of loop, we would have found all non-subsumed tuples. Now subtract them from total tuples to get all subsumed tuples.

Any answer with all possible comparisons and $O(n^2)$ gets minimum credit (4-8). Solutions which do not find correct subsumed tuples, but are $O(n \log n)$ will also receive minimum credit. An answer with worst case $O(n^2)$ but uses some kind of sorting and trying to find max-tuple and use it for pruning will receive 10-15 points. An partial answers which uses sorting and pruning with max-tuple will receive between 16-24 points. A solution with minor mistakes (like not deriving complexity) will get 25-30 points.
4. Graded by Manna

Again, this was a throwaway problem. While doing DFS or BFS from any vertex as root. Do the following recursively: for every edge \((u, v)\), put \(v\) in the opposite set of the parent vertex \(u\) (the root can be put in either \(V_1\) or \(V_2\) arbitrarily). Anytime you find that parent and child end up being on the same set, you terminate saying that the graph is not bipartite. You can also use the existing DFS technique of assigning discovery timestamps: anytime you find an edge \((u, v)\) where it is a forward edge or back edge or a cross edge, compare the \(d[u]\) and \(d[v]\). If they are both odd or both even, then the graph is not bipartite. If that never happens, then the graph is bipartite. The reasoning should be obvious – you can just put the vertices with odd \(d[]'s\) in one side and vertices with even \(d[]'s\) on the other side, and we know that there is no edge that covers vertices with both odd discovery time or both even discovery time (no edge within the set).

And though I hate to say this: many of you got it wrong and were penalized heavily. I am including some of the answers that were incorrect:

- **Find if the given graph is a tree / if it contains a cycle / if there is a back or cross edge:**
  All trees are by definition bipartite; because there is only one way to reach a vertex from another. Unfortunately, the graph may not be a tree yet it can be bipartite. For example, a quadrilateral, which is not a tree and contains a cycle, is bipartite.

- **Do a BFS and find if there exists an edge between grey nodes (assuming grey nodes are those that are inside the queue or the one being explored) - if there is, it is not bipartite.** Does not work - try with a quadrilateral.

- **Do a DFS and find if there exists an edge between grey nodes - if there is, it is not bipartite.** Again, does not work - try with a quadrilateral.