In the previous lecture, we introduced the notion of Group Testing. In group testing, we are given a pair of integers $(d, n)$ such that $d \leq n$. We need to compute an unknown vector $x \in \{0, 1\}^n$, where $|x| \leq d$ using as few number of tests $t$ as possible. There are two kinds of tests. The first method is called adaptive testing. For adaptive testing $t^a(d, n)$ is used to denote the minimum number of adaptive tests needed for a pair $(d, n)$. Adaptive tests are done by performing a test and then basing the next test to perform off of the results. The second method is called non-adaptive testing. For non-adaptive testing $t(d, n)$ is used to denote the minimum number of non-adaptive tests needed for a pair $(d, n)$. Non-adaptive tests are done by fixing all tests apriori.

Group testing can be formalized as follows:

- **Input**: $(d, n)$ such that $d \leq n$ and (unknown) $x \in \{0, 1\}^n$
- **Tests**: Query/test any subset $S \subseteq [n]$. The answer given is $\bigvee_{i \in S} x_i$. Note that the combination of all tests can be represented as a matrix $T$, where $a_{j,k}$ is 1 if for test $j$, $k \in S$. Then $T \times x = r$ where $r$ is the result of the matrix multiplication where multiplication is logical AND and addition is logical OR. After performing the $j$th test, the value $r_j$ is obtained.
- **Output**: $x$

From discussion in last lecture and the definition of adaptive and non-adaptive tests we have

$$1 \leq t^a(d, n) \leq t(d, n) \leq n$$

The reason for $t^a(d, n) \leq t(d, n)$ is due to the fact that any non-adaptive test can be performed by an adaptive test by running all of the tests in the first step of the adaptive test. Adaptive tests can be faster than non-adaptive tests since the test can be changed after certain things are discovered.

In todays lecture, we will prove sharper bounds on $t^a(d, n)$ and $t(d, n)$.

## 1 Lower Bound on $t^a(d, n)$

**Proposition 1.1.** $t^a(d, n) \geq d \log\frac{n}{d}$

Fix any valid group testing scheme $t^a(d, n)$ with $t$ tests. Observe that if $x \neq y \in \{0, 1\}^n$, with $|x|, |y| \leq d$ then $r(x) \neq r(y)$, where $r(x)$ denotes the result vector for running the tests on $x$ and similarly for $r(y)$. The reason for this is because two valid inputs cannot give the same result. If this were the case and the results of the tests gave $r(x) = r(y) = r$ then it would not be possible to obtain both $x$ and $y$. This fact gives us the following:
Total number of distinct test results = $Vol_2(d, n)$

The number of possible distinct $t$-bit vectors is $2^t$, and since $2^t \geq Vol_2(d, n)$ it implies $t \geq \log Vol_2(d, n)$.

Recall that $Vol_2(d, n) \geq \binom{n}{d} \geq (\frac{n}{d})^d$ so $t \geq d \log Vol_2(d, n)$. Therefore, since $t^a(d, n) \leq t(d, n)$, no matter which scheme is used it cannot perform better (use fewer) than $d \log \frac{n}{d}$ tests.

## 2 Upper Bound on $t^a(d, n)$

**Proposition 2.1.** $t^a(d, n) \leq O(d \log n)$

The following is an example of an $O(d \log n)$ adaptive group testing scheme. The idea of the overall algorithm is to use a binary search and repeat until at most $d$ values are found or no more values remain to be found.

**Toy Problem:** Give a scheme that uses $O(\log n)$ adaptive tests to figure out ONE $i$ such that $x_i = 1$ (otherwise report $|x| = 0$).

**Warmup:** Query $[n]$ to check if $|x| = 0$ (i.e. check if $\bigvee_{i=1}^{n} x_i = 0$.)

**General Case:** Split $[n]$ into two equal halves. Query the first half and if the result is 1 then recurse on that set by splitting those indices in half and repeating this process. If the query on the first half is not 1 then query the second half (note that if the query of the entire set was performed then querying the second half is redundant since it would be known there is a 1 here). If querying the second half of the indices gives a result of 0 then report there is no 1 exists in this section. Continue this process on the subset containing a 1 until either the set only contains one element or no 1 is found. If a 1 is found and the set contains only one element report that index as being valued 1.

This will take $2 \lceil \log n \rceil$ or, provided the first test is performed querying the whole set, $\lfloor \log n \rfloor + 1$ queries given that if one half is 0 it implies the other half 1.

**General Algorithm:** Let $S = [n]$

1. Find one $i \in S$ such that $x_i = 1$ using the algorithm described in the General Case above.

2. Let $S = S \setminus \{i\}$ and then repeat the algorithm on $S$. If the first step reports there are no values left then stop. Also stop after $d$ iterations.

This algorithm will run for $d$ iterations of the first algorithm, giving an overall runtime of $O(d \log n)$. Since this is a general algorithm for an adaptive test any adaptive test is bounded by this, so $t^a(n, d) \leq O(d \log n)$.

Note that the algorithm above is inherently adaptive and thus the argument above does not give an upper bound for $t(d, n)$. It is impossible to have this as a lower bound for $t(d, n)$ since there is a known (not proved in class) bound $t(d, n) \geq \Omega(\frac{d^2 \log n}{\log d})$. 

2
3 Upper Bound on $t(1, n)$

**Proposition 3.1.** $t(1, n) \leq \lceil \log n \rceil$

The group test matrix that is the parity check matrix for $[2^m - 1, 2^m - m - 1, 3]_2$, i.e. $H_m$ where the $i$-th column is the binary representation of $i$, will work for any unknown $x$ where $|x| \leq 1$. This works because when performing $H_m x = r$, if $|x| \leq 1$ then $r$ will correspond to the binary representation of $i$. Therefore the lower bound for $t(1, n)$ is $\lceil \log n \rceil$. If $n \neq 2^m - 1$ for some $m$, the matrix $H_m$ corresponding to the $m$ such that $2^{m-1} - 1 < n < 2^m - 1$ can be used by adding 0s to the end of $x$. By doing this, decoding is "trivial" for both cases since the binary representation is given for the location. So the number of tests is $\lceil \log n \rceil$. Therefore since the number of tests is an integer, we have $t(1, n)$ is upper-bounded by $\lceil \log n \rceil$.

Recall the lowerbound for $t(d, n)$ is $d \log \frac{n}{d}$, so with $d = 1$ we have $\log n \leq t(1, n) \leq \log n$, so $t(1, n) = \lceil \log n \rceil$ is an efficient code. Such tight bounds are not known for general $d$. 