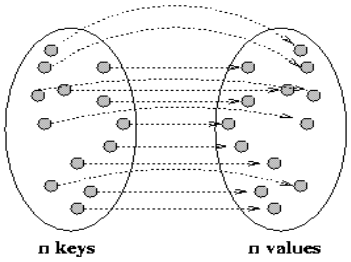
1 Hashing

1.1 On Hashing

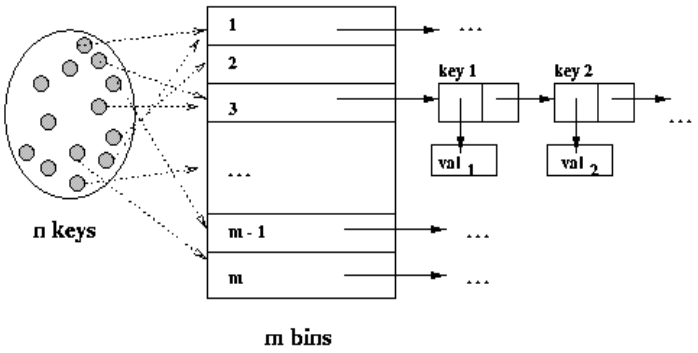
We are given the task of storing a dictionary of word-definition pairs – i.e., a set of n $(key, value)$ pairs. We could store these pairs in a linked list, but this requires a $O(n)$ access time.



We can choose a hash table as an alternative storage structure. Instead of keeping the n keys in order with their n values, we divide the keys into m slots, or bins, using a hash function $f : f(key_k) \rightarrow bin_s$, $0 < s \leq m$. Within each bin we maintain a data structure, such as a list L_k, that allows us to search within the keys which hash to bin k.

1.1.1 The advantage of hashing over binary search trees

Hash tables are efficient when it comes to random access. Unlike in lists, where we do not know ahead of time where we will find a value, the hash function points to the bin where the value is expected to be.



In the good case, each bin contains n/m keys, and we can expect to search in $\approx n/m = \alpha$, where α is the load factor. Thus, the expected search time is $O(1)$ by assumption.

In this good case, the length of the lists holding the values that have collided, or hashed to the same bin, are α. However, in the worst case (when all the keys hash to the same bin), the search time is $O(n)$!

We thus consider **two strategies when the worst case is unacceptably bad:**

1. We assume that the worst case won't occur, or that it is very unlikely. In the case of hash tables, this makes an assumption about the input, i.e., that it is evenly distributed. **This is the study of average case analysis.**

2. Use randomness inside the algorithm. **This is the study of randomized algorithms.**

In this class, we will focus on the second of these strategies.

1.1.2 Probability theory

Probability distribution: Over a finite space Ω, we consider the function $p : \Omega \rightarrow [0, 1]$ with the property

$$\sum_{x \in \Omega} p(x) = 1$$

.

(i.e., the space Ω is a finite collection of numbers whose sum is 1.)
For any event $E \subseteq \Omega$, the probability that E occurs, or $Pr[E]$ is $\sum_{x \in E} p(x)$. It follows that $0 \leq Pr[E] \leq 1$.
We consider a random variable function $X : \Omega \rightarrow R$.
We use as an example a coin toss. In this case,

$$\Omega = \{outcomes(5\ coin\ tosses)\} = \{HHHHH, HHHHT, \ldots\}$$

Expectation: $E[X] = \sum_{x \in \Omega} p(x) \times X(x)$ (This is sometimes referred to as a weighted average since $\sum p(x) = 1$.)

THEOREM: Linearity of expectation: If x, y are random variables and c is a constant, then

$$E[x + y] = E[x] + E[y]$$
$$E[cx] = cE[x]$$

DEFINITION: We define an **indicator random variable** $\mathcal{I}(E) : \Omega \rightarrow R$ for an event $E \subseteq \Omega$:

$$\mathcal{I}(E) = \begin{cases} 1 & \text{if the event occurs} \\ 0 & \text{otherwise} \end{cases}$$

More formally, $\mathcal{I}(E) : \Omega \rightarrow R$ is defined as

$$\mathcal{I}(E)(x) = \begin{cases} 1 & \text{if } x \in E \\ 0 & \text{otherwise} \end{cases}$$

We note that if $\mathcal{I}$ is an indicator random variable, then

$$E[\mathcal{I}] \ = \ Pr[\mathcal{I} = 1] \ = \ Pr[the\ corresponding\ event\ occurs]$$

.

PROOF: $E(\mathcal{I}) = 0(Pr[\mathcal{I} = 0]) + 1(Pr[\mathcal{I} = 1]) = 0 + Pr[\mathcal{I} = 1]$

1.2 Universal Hashing

[CLRS: 1.3.3; MR 8.4.1-4]
Note: We will be using the notation and reference to "universal" and "two universal" of the CLRS presentation.

For this presentation, we assume that $\alpha = O(1)$.

In designing an efficient hashing scheme, we know that we can't just use a fixed hashing algorithm, because it would be possible for our adversary to design input that would put our search algorithm in its worst case (i.e., all the input keys could hash to the same location).

We thus consider a family of functions as part of our hashing scheme. When we need to construct a new hash table, we choose one function of this family at random, and use it in the construction. (We also remember which function we used, or else we would not be able to search the table after construction.) Given a family $\mathcal{H}$ of hash functions from M to N, $M = \{0, 1, \ldots, m - 1\}$, we pick one function h from $\mathcal{H}$ *uniformly at random.*

DEFINITION: $\mathcal{H}$ is called **universal** if $\forall$ keys k, l, $k \neq l$, $Pr_{h \in \mathcal{H}}[h(k) = h(l)] \leq 1/m$.

This goes to say that the number of hash functions $h \in \mathcal{H}$for which $h(k) = h(l)$ is at most $|\mathcal{H}|/m$, or, with a randomly chosen h the chance of a collision is no more than if $h(k)$ and $h(l)$ were randomly chosen from the values $\{0,1,\ldots,m\text{-}1\}$.

DEFINITION: We say that $\mathcal{H}$ is **two-universal** if $\forall$ x_1, $x_2 \in M$, $x_1 \neq x_2$, any $y_1, y_2 \in N$, and h chosen uniformly at random from $\mathcal{H}$,

$$Pr[h(x_1) = y_1 \text{ and } h(x_2) = y_2] = 1/n^2$$

We note that the first form of universality regards the probability that two keys collide; the second form concerns the probability that two keys hash to two certain values (which may or may not constitute a collision).

THEOREM: Using a universal hash function family gives $E[search\ time] \leq 1 + \alpha$.

PROOF: We define two indicator random variables:

$$X_{kl} = \mathcal{I}\{h(k) = h(l)\}$$

which is the indicator random variable for the event that k and l collide;

$$Y_k = \sum_{l \in M; l \neq k} X_{kl}$$

which is the number of other keys that hash into the same slot as k ($h(k)$).

$$\begin{aligned} E[Y_k] &= \sum_{l \in M; l \neq k} E[X_{kl}] \\ &= \sum_{l \in M; l \neq k} Pr[X_{kl} = 1] \\ &= \sum_{l \in M; l \neq k} Pr[h(k) = h(l)] \\ &\leq \sum_{l \in M; l \neq k} 1/m(by\ definition) \\ &= 1/m(the\ number\ of\ keys\ \neq k) \\ &\leq 1/m(n) \\ &= \alpha \end{aligned}$$

We note for Y_k that k is from a larger universe of possible keys; k doesn't have to already be in the table. If it *is* in the table, we don't want to count the collision of k with itself. So, we bound Y_k by α.

Search time: the length of the list at location $h(k)$

$$= \begin{cases} Y_k & \text{if } k \notin the\ table \\ Y_{k+1} & \text{if } k \in the\ table \end{cases}$$

$$\begin{aligned} E[search\ time] &\leq E[Y_k + 1] \\ &= E[Y_k] + 1 \\ &\leq \alpha + 1 \end{aligned}$$

1.3 Designing a Universal Hash Family

For this discussion, we will consider the keys of our hash functions as integers. We let p be some prime $>$ any key. We define the hash family $\mathcal{H}$ which consists of functions $\mathcal{H}_{a.b} : \{0, 1, \ldots, p-1\} \rightarrow \{0, 1, \ldots, m-1\}$ for all a and b

$$h_{a,b}(k) = ((ax + b_m od\ p) mod\ m$$

THEOREM: $\{h_{a,b} : a \in \{1, \ldots, p-1\}, b \in \{0, p-1\}$ is universal.

PROOF: We note that a can't be 0, or all keys would map to the same slot. We note also that a function $mod\ p$ is a linear operation over some field {0,...,p-1}.

We want to prove that the probability of a collision is bounded by a value. There are two places for a possible collision to occur: $mod\ p$ and $mod\ m$.

We show first that there cannot be a collision at $mod\ p$.

- We suppose that $k \neq l$ and $(ak + b) mod\ p\ = (al + b) mod\ p$. So, p divides $a(k - 1)$, since p does not divide a, because $|a| < p$, $a \neq 0$.
- Because p is prime, we cannot factor it, and it must thus divide $(k - l)$. However, $|k - l| < P$, $\neq 0$, so p can't divide $(k - l)$ either.
- This yields a contradiction; distinct keys thus cannot collide at the level of $mod\ p$.

Next, we suppose that $k \neq l$ are two distinct keys. Let

(1) $r = (ak + b) mod\ p$ **(2)** $s = (al + b) mod\ p$.

Then, $r \neq s$. We show that r and s can take on all possible values in $\{0, \ldots, p-1\} \times \{0, \ldots, p-1\}$ except for those where $r = s$. As an aside, we note that:

$$\left.\begin{array}{l} r_1 = (ak_b) mod\ p \\ s_1 = (al_b) mod\ p \end{array}\right\}\ has\ a\ solution\ in\ a, b.$$

Two linear equations have a solution when

$$\begin{array}{rcl} k \times a + 1 \times b & = & r \\ l \times a + 1 \times b & = & s \end{array}, \left| \begin{array}{cc} k & 1 \\ l & 1 \end{array} \right| \neq 0$$

The determinant is easy to evaluate:

$$k - l \neq 0$$
$$k \neq l$$

(We started with the second condition.)

Everything is still acceptable $mod\ p$ because p is prime; working mod a prime is the same as working in real numbers.

If the determinant is non-zero, there is a solution and it is a unique solution.

By the theory of linear equations, $\forall (r, s), r \neq s, \exists$ a unique (a, b) such that **(1)** and **(2)** hold.

This means that there is a one-to-one correspondance between (a, b) and (r, s); choosing $h_{a,b}$ at random is equivalent to choosing (r, s) at random, subject to the condition that $r \neq s$.

Thus, $Pr[collision] = Pr[r\ mod\ m = s\ mod\ m]$ (where $r, s \leftarrow 0, \ldots, p-1$ and $r \neq s$). If we remove the requirement that $r \neq s$, we find that

$$\begin{array}{rl} Pr[collision] & = Pr[r\ mod\ m = s\ mod\ m] - 1/p \\ & = Pr[s\ mod\ m = some\ particular\ value - 1/p \end{array}$$

Thus we see that the the probability of a collision is bounded:

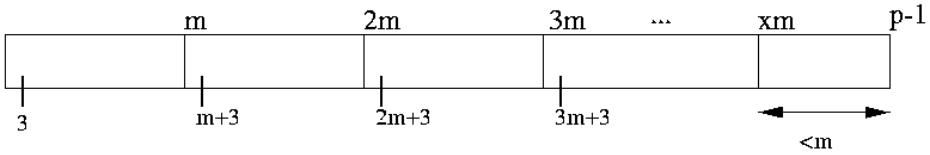
$$\begin{aligned} Pr[collision] = & \quad Pr_{r,s\leftarrow\{0,\ldots,p-1\},r\neq s}[r \bmod m = s \bmod m] \\ = & \quad Pr_{s\leftarrow\{0,\ldots,p-1\},r\neq s}[r \bmod m = s \bmod m] \end{aligned}$$

(Which is the same probability that an s chosen at random results in a collision.)

$$= Pr_{s\leftarrow\{0,\ldots,p-1\}}[r \bmod m = s \bmod m] - 1/p$$

(the $1/p$ represents an additional value of s, where $s = r$)

$$= Pr_{s\leftarrow\{0,\ldots,p-1\}}[s \bmod m = some\ particular\ value] - 1/p$$



(Say you pick the number 3– we can have a collision by choosing 3, $m + 3$, $2m + 3$, $3m + 3$, etc.)

We note that the final segment of this drawing is of size $< m$, as p is a prime and does not divide by m. Since it is thus possible that it will not contain the value we're looking for, we take the ceiling of this value.

$$Pr[collision] \leq \left\lceil \frac{p-1}{m} \right\rceil 1/p - 1/p \leq 1/m$$

1.4 Perfect Hashing

[CLRS 11.5; MR 8.5]

What if we could make the search time $O(1)$ in the *worst* case? We can accomplish this using a two-level hash table in which the second hash function is collision-free. How do we achieve a simple hash function that is collision free? We use a universal hash family with a table size of n^2, according to the scheme discovered by Fredman, Komlos, and Szemeredi (1984-1986).

CLAIM: Let h be chosen uniformly at random from a universal hash family mapping keys to $0, \ldots, n^2 - 1$. Then, if we hash n keys using h, $Pr[\exists collision] < 1/2$.

To prove this, we digress for a moment to note Markov's inequality: if $x \geq 0$ is a random variable, then $Pr[x \geq t] \leq \frac{E[x]}{t}$ for any $t > 0$.

Intuitively, this tells us that a random variable's probability to take on a value decreases the further that value strays from the variable's expected value.

PROOF: Of Markov's Inequality (for finite probability spaces)

Let $x_1, \ldots, x_r$ be the possible values of x. Then,

$$\begin{aligned} E[x] & = \textstyle\sum_{i=1}^{r} x_i\ Pr[x = x_i] \\ & = \textstyle\sum_{x_i<t} x_i\ Pr[x = x_i] + \sum_{x_x\geq t} x_i\ Pr[x = x_i] \\ & \geq \textstyle\sum_{x_x\geq t} x_i\ Pr[x = x_i] (because\ x \geq 0) \\ & \geq \textstyle\sum_{x_x\geq t} t\ Pr[x = x_i] \\ & = t\ Pr[x \geq t] \end{aligned}$$

Dividing by t,

$$\frac{E[x]}{t} \geq Pr[x \geq t]$$

Now, we prove the above claim concerning the probability of a collision for universal hash functions and a hashed space of size n^2.

PROOF: Let $x = the\ number\ of\ collisions.$

$$\begin{aligned} E[x] &= \sum_{all\ pairs\ (k,l)} Pr[h(k) = h(l)] \\ &= \sum_{all\ pairs\ (k,l)} 1/n^2 \\ &= \binom{n}{2} 1/n^2 = \frac{n-1}{2n} < 1/2 \end{aligned}$$

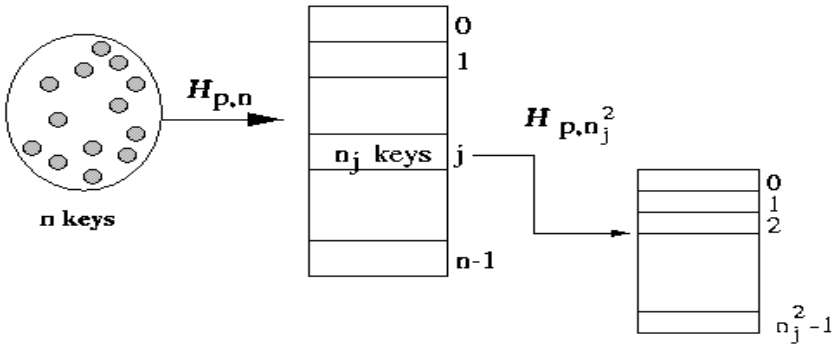
So we know that the expected number of collisions is $1/2$. We want to say something about the probability of a collision. Applying Markov's inequality:

$$\begin{aligned} Pr[\exists\ collision] &= Pr[x \geq 1] \\ &\leq \tfrac{E[x]}{1} < 1/2 \end{aligned}$$

The idea here is that if we have an initial set of data, given that it is more likely than not that there will be no collisions, it is quick to find within a few random guesses a hashing function that gives no collisions.

1.5 Optimized Space for Perfect Hashing

We showed above that picking a hash function uniformly at random from $\mathcal{H}_{p,n^2}$ gives *no collisions* with probability $> 1/2$. However, we should ideally only use $O(n)$ space to store n keys. Our second try for a bound on this space used to store the hash table will be $n + O(n)$.



The size of this data structure will be $n + {n_1}^2 + {n_2}^2 + \cdots + {n_n}^2$, or $n + \sum_{j=1}^{n} {n_j}^2$. We note that the number of keys that hash to a particular bin (i.e., the values of n_j in the preceeding sum) is determined by the hash function that is chosen.

We want to bound $E[n + \sum_{j=1}^{n} n_j^2] = n + E[\sum_{j=1}^{n} n_j^2]$. We do this with an algebraic trick: $a^2 = a + 2\binom{a}{2}$.

Using this trick, we find $\sum_{j=1}^{n} n_j^2 = \sum_{j=1}^{n} n + 2\sum_{j=1}^{n} \binom{n_j}{2} = 2n\ times\ number\ of\ collisions\ in\ the\ hash\ function$).

$$\begin{aligned} E[\textstyle\sum_{j=1}^{n} n_j^2] = &\quad n \ + \ 2 \ \times \ E[\#collisions\ of\ primary\ hash\ function] \\ \leq &\quad n \ + \ 2 \ \times \ \left(\begin{array}{c} n_j \\ 2 \end{array} \right) \ \times \ 1/n \\ = &\quad n \ + \ (n-1) = 2n-1 \\ E[size\ of\ data\ structure] = &\quad E[n + \textstyle\sum_{j=1}^{n} n_j^2] \leq n + (2n-1) < 3n \end{aligned}$$

How likely is it that we'll actually get the expected size of this data structure when implemented? To discover this, we apply Markov's Inequality.

$$Pr[size \geq 6n] \leq \frac{E[size]}{6n} < 1/2$$

We note that to use this, we must obey the condition that the random variable be > 0. This is true in this case, since the size of the structure will be greater than zero.

It is actually possible to come up with a slightly tighter bound:

$$\begin{array}{ll} E[\sum_{j=1}^{n} {n_j}^2] \leq 2n-1 & < 2n \\ Pr[\sum_{j=1}^{n} {n_j}^2 \geq 4n] < \frac{2n}{4} & = 1/2 \\ Pr[size \geq 5n] & < 1/2 \end{array}$$

1.6 Algorithm and Run Time for Space-Optimized Perfect Hashing

We consider the algorithm by which one constructs a hash table in this way.

1. Choose a function from $\mathcal{H}_{p,n}$ uniformly at random.
2. Compute all h_j and sets k_j of keys hashing to slot j by actually performing the hashing.
3. If $\sum_{j=1}^{n} {n_j}^2 > 4$, go to step 1.
4. For $j = 0\ to\ n-1$,
   (a) Choose a function h_j from $\mathcal{H}_{p,n_j}$ uniformly at random.
   (b) Hash the keys in k_j using h_j.
   (c) If there is a collision in $\mathcal{H}$, go to step 4.1.

The expected run time is $O(n)$ for steps 1-3 and $\sum_{j=1}^{n}(O({n_j}^2))$ for steps 4.1-4.3, for a total of $\leq O(n)$.

1.7 Refining the Space Optimization for Perfect Hashing

(The notation used in this section is similar to that of the paper of which it is a presentation, *Storing a Sparse Table with O(1) Worst Case Access Time*, by Fredman, Komlos, and Szemeredi.)

Given $W \subseteq U$ with $|W| = r$, $k \in U$, and $s \geq r$, let $B(s, W, k, j)$ be the number of times that the value j is achieved by the function $x \rightarrow (lk \ mod\ p) mod\ s$ when x is restricted to W. This refers to the number of values in the bin, not the number of collisions.

We note that k plays the same role as a in $h_{a,b}$, and that s plays the same role as m from $h_{a,b}$. The range of the function is from $\{0 \ldots s-2\}$; it is not universal.

LEMMA 1: Given the above, there exists a $k \in U$ such that $\sum_{j=1}^{s} \left(\begin{array}{c} B(s,W,k,j) \\ 2 \end{array} \right) < r^2/s$

$$\sum_{k=1}^{p-1} \sum_{j=1}^{s} \left(\begin{array}{c} B(s,W,k,j) \\ 2 \end{array} \right) < \frac{(p-1)r^2}{s} \tag{1}$$

Equation 1 expresses the number of (kx, y) with $x, y \in W, x \neq y, 1 \leq k \leq p$, such that $(kx \ mod \ p) \ mod \ s > (ky \ mod \ p) mod \ s$; i.e., the number of actual collisions.

The contribution of x, y to this quantity is at most the number of k for which

$$k(x-y) mod \ p \in s, 2s, 3s, \ldots, p-s, p-2s, p-3s, \ldots \tag{2}$$

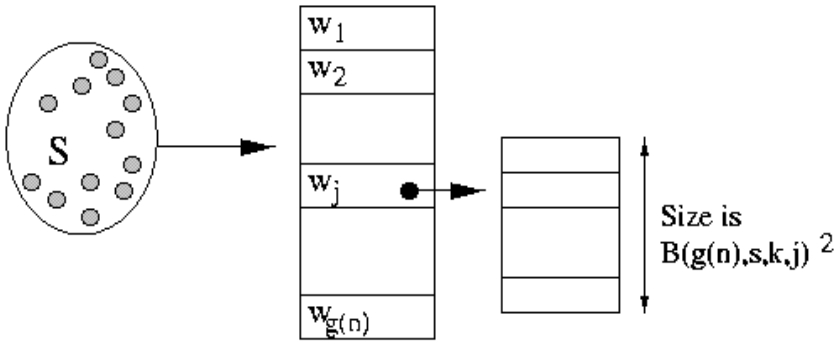
Because $x - y$ has the multiplicative inverse $mod \ p$, the number of k satisfying 2 is $\leq \frac{2(p-1)}{s}$.
(This is similar to the argument we saw in class.)
When we sum over the $\begin{pmatrix} r \\ 2 \end{pmatrix}$ possible choices for x, y, we get $\begin{pmatrix} r \\ 2 \end{pmatrix} \frac{2(p-1)}{s} = \frac{(p-1)(r(r-1))}{s}$.
We conclude that the sum in (1) is bounded by $\frac{(p-1)r^2}{s}$.
Using Lemma 1, we let $W = S, s = g(n), r = n$. We find that for some k,

$$\sum_{j=1}^{O(n)} \begin{pmatrix} B(s, W, k, j) \\ 2 \end{pmatrix} = O\left(\frac{n^2}{g(n)}\right) \tag{3}$$

We note a quick algebraic trick: $x^2 = O(\begin{pmatrix} x \\ 2 \end{pmatrix})$ for $x \geq 2$, and use it to find that

$$\sum\nolimits' (B(g(n), s, k, j)^2 = O\left(\frac{n^2}{g(n)}\right) \tag{4}$$

Where $\sum'$ denotes the sum over j, as j goes from 1 to $g(n)$ and $(B(g(n), s, k, j) \geq 2$.
Since $g(n)$ will be chosen such that $lim \frac{n}{g(n)} = 0$, 4 implies that the total space required to resolve those blocks having 2 or more values is $O(n)$.



W_j can be a pointer to a secondary table or an immediate value. It's hard to know which is which, so we maintain another data structure to keep track of this information. It contains "tag bits" that identify whether a bin W_j in the first-level table contains an immediate value or a pointer to a secondary-level table.

These tag bits will take up $O(\frac{n}{log \ m})$ space, or $o(n)$ words. (These are single binary bits, and words are of length $log \ m$ bits.)

We need an auxiliary data structure to determine also whether a block W_j is nonempty. Given T, the memory used to store the table, we note T', which is the first address of the cells $W_1 \ldots W_{g(n)}$ which are allocated consecutively in order in memory.

On the interval $I = [1, g(n)]$ we partition $\frac{n^2}{g(n)}$ subintervals of size $(\frac{g(n)}{n})^2$.

With each subinterval σ of I, we associate a base address $BA[\sigma]$. This is the base address of the location preceding the cells in T' associated with the non-empty $W_j, j \in \sigma$. These base addresses are stored in a table of size $\frac{n^2}{g(n)} = O(n)$.

We create a second table of addresses, $A[j], j \in I$, that is used to store the offsets. $A[j] = 0$ if $W_j = \emptyset$; otherwise, $BA[j] + A[j]$ is the address in T' associated with W_j for $j \in \sigma$.

How much space does this new table take? Since $A[j]$ assumes at most $(\frac{g(n)}{n})^2$ possible values, all $A[j], j \in I$ can be packed into space

$$O\left(\frac{g(n)log(\frac{g(n)}{n})}{log\ m}\right) \tag{5}$$

In 5, $g(n)$ is the number of blocks in W_i, $log\left(\frac{g(n)}{n}\right)$ comes from $log\left(\frac{g(n)}{n}\right)^2$ and the fact that $log(x^2) = 2\ log\ x$ and expresses the number of bits used to store $A[j]$, and $log\ m$ is the number of bits that can fit in a word, $m > n$.

We choose $g(n) = n\sqrt{log\ n}$:

$$\begin{aligned} & O\left(\frac{g(n)log(\frac{g(n)}{n})}{log\ n}\right) \\ = \quad & O\left(\frac{m\sqrt{log\ n}\ log(\sqrt{log\ n})}{log\ n}\right) \\ = \quad & O\left(\frac{n \times \frac{1}{2} log\ log\ n}{\sqrt{log\ n}}\right) \end{aligned}$$

We note that the denominator grows faster than the numerator; thus, the total space used is $O(n)$.