Jone's & Kelly (Continued..)

\[(lo, hi) - \text{lookup (p)}\]
\[\text{assert (lo} \leq p < hi)\]
\[*p\]

**Evaluation Criteria**

1. **Overhead**: How much slower will the program run?
   Jone's & Kelly → 10 x slower

   Use of hash table is possible but it has some problem.
   So, J&K uses tree.

![Tree Diagram]

2. **Memory Overhead**: There is separate node for each allocation but memory is not a big problem.

3. **Compilatiojn Speed**: Nobody Cares 😊

4. **Effectiveness**:
   a. Temporal Memory Safety
   b. Intra-structure overflows
Temporal Memory Safety

```c
int *p = malloc (...);
...
...
free (p);
q = malloc (...);
*p = 5;
```

```c
struct foo {
    int a;
    float b;
};
struct foo *p = malloc (...);
int *q;
q = &p → a ;
q = q + 1 ;
*q = 37
```

In the end, q will point to float number. Being a int *, so that’s a problem.

5. **False Positives**: (May have small number of false positives)

```c
int A [10] ;
insert (A, 10) ;
int *p = A ;
for (i=0; i<10; i++) {
    (lo, hi) = lookup (p);
    assert (lo <= p < hi);
    *p = 0 ;
    (lo, hi) = lookup (p);
    assert (lo <= p+1 < hi);
    p = p+1 ;
}
```

```
i = 0  p = A+1
i = 1  p = A+2
...
...
i = 9  p = A+10  //Out of Bounds
```

```
1 extra pad byte
(to overcome this problem)
```
The reason behind checking every pointer arithmetic operation is, That it may go out of bounds and point to other object sometimes. For example:
\[ p = p + 10 \]
\[ *p = 0 \]
To check that every time a pointer points to its own object.

6. **Separate Compilation Support** : J&K are great at this, because their method does not care about previous or next code.

7. **Linking transformed and untransformed code** :
If, library/untransformed code is not buggy, still then, Memory allocation is untransformed code could be a problem.

8. **No change Code**: It means changing code when problem cannot be found. (J&K → no code change)

Out of above described, there are 6 major concerns.

- Speed Overhead
- False Positives
- Effectiveness
- Separate Compilation Support
- Linking untransformed and transformed code
- No change code

**Baggy Bounds Checking**

- Some similarities to J&K.
- Bounds kept for objects
  - Pointer can never go out of bounds.
- Efficient lookups
- Efficient checks
- Other optimizations

Main trick is to enable lookup via hashing.

\[
\text{BNDS} \left[\frac{2^{32}}{16}\right] \quad \text{// 2}^{28} \text{ bytes}
\]

\[
\text{BNDS} \left[\frac{P}{16}\right] = \log_2 (\text{size of memory allocated around } P)
\]

- Memory will all be in form of \(2^x\)
- Also aligned to \(2^x\)

![Diagram of Baggy Bounds Checking](image-url)