8. Custom Block Allocation

An application with heavy use of malloc() and free() for a particular data structure may obtain a significant performance increase using block allocation and caching.

For example, data nodes could be allocated in blocks of 1000, so then to allocate 1,000,000 nodes malloc() would be called only 1000 times.

And instead of using free(), unneeded nodes could be saved for later reuse.

Example implementation:

```
void *emalloc( size_t nbytes) { // malloc() with exit on error
  void *p = malloc(nbytes); if(!p) { fprintf( stderr, "malloc() failed\n"); exit(1); } return p;
}

typedef struct node { double x; int y; /* ... data ... */ struct node *next; } Node;

#if BLOCK
static Node *head = 0;
static int block = 1000; // number of nodes to allocate at one time
Node *new_node( void) {
  if( !head) { // allocate a block of nodes
    head = emalloc(block*sizeof(Node)); Node *last = head+block-1;
    for( Node *p = head; p < last; ++p) p->next = p+1; // link the nodes
    last->next = 0;
  }
  Node *n = head; head = head->next; return n; // return head of list
}
void free_node( Node *n) {
  n->next = head; head = n; // insert at head of list
}
#else
#define new_node()   emalloc(sizeof(Node))
#define free_node(n) free(n)
#endif
```