***Characteristics of a Javascript Array***

1. **Dynamic** - Arrays in Javascript can grow dynamically .push
2. Can be **sparse** - For e.g. array[50000] = 2;
3. Can be **dense** - For e.g. array = [1, 2, 3, 4, 5]

*Dynamic & Sparse Array Implementation:*  
  
When the runtime detects that the array is Sparse, it is implemented in a similar way to an object.

So instead of maintaining a contiguous array, a key/value map is built.

In case of an array since the indexes are integers, this map can be efficiently implemented using a balanced Binary Search Tree or a similar structure. The performance of a sparse array is likely similar to the performance of an object.

The arrays are initialized with Flat arrays and are then later replaced with Sparse array map if necessary.

1. Never use an object, when you need an array  
   2. Never initialize arrays backwards (this will force create sparse array which is less efficient than dense arrays)

When working with JavaScript arrays two independent assumptions are prevalent:

1. They behave like C/Java arrays. Iteration over the contents and accessing data by index is fast — deleting entries and extending the size is expensive.
2. They behave like JavaScript objects. Iteration over the contents and accessing data by index is slow — deleting entries and extending the size is fast.

In JavaScript arrays are a special case of objects.

In fact they inherit from Object.prototype, which also explains why typeof([]) == “object”.

The keys of the object are positive integers and in addition the length property is always updated to contain the largest index + 1. This supports the second assumption (array == object).

Since arrays are just special cases of objects we thought that rewriting the code to use objects should speed up deletions but maintains all other performance characteristics:

listeners[id] = {  
 callback: callback,  
 context: self  
}

The somewhat unexpected result was that while deletions became faster the overall performance became much worse.

So while Arrays and Objects are conceptual almost the same, most JavaScript engines treat them very differently.

var ar = [];  
for (var i = 0; i < 500000; i++) {  
 ar[i] = i;  
};

var map = {};  
for (var i = 0; i < 500000; i++) {  
 map[i] = i;  
};  
These two loops are almost identical but the first one can be up to **200 times** faster (Firefox 3.6) than the second one.

So we know that arrays are treated differently by the JavaScript engines but under which circumstances do the optimizations kick in? To test this we kept the first loop but added the following statements before the loop:

* ar[10] = 10; // use the array as a small sparse array
* ar[50000000] = 10; // use the array as a huge sparse array
* ar.monkey = 10; // treat the array as an object
* **WebbKit:** In WebKit there is a huge difference between objects and arrays. Further it is the only one which can optimize the huge sparse array case. WebKit on the other hand doesn’t like the reverse case at all. To me this looks like a bug in the implementation.
* **Chrome 5 (V8):** The V8 JavaScript engine of Chrome does an amazing job. With the exception of the huge sparse array case, all tests took almost the same time. From a performance point of view there is almost no difference between objects and arrays.
* **FireFox:** We have tested two FireFox versions. Firefox 3.0 with the old SpiderMonkey engine and FireFox 3.6 with the tracing JIT TraceMonkey. The characteristics of both are very similar with FireFox 3.6 being a lot faster. If the array is used purely as array, Firefox 3.6 is the absolute winner. In all other cases performance degrades to the much slower object case.
* If in doubt, prefer arrays over objects.
* Never treat arrays as objects.
* Never initialize arrays backwards.
* Sparse arrays behave (performance-wise) like objects.
* Unusual array usage has a cost.